UNIVERSIDADE FEDERAL DE SANTA CATARINA

CURSO DE PÓS-GRADUAÇÃO EM CIÊNCIA DA COMPUTAÇÃO

IMPLEMENTANDO APLICAÇÕES DISTRIBUÍDAS UTILIZANDO CORBA E DCOM: UM ESTUDO DE CASO VOLTADO À ÁREA DE BANCO DE DADOS

ANAMÉLIA CONTENTE DE SOUZA

Florianópolis

Novembro de 1999

UNIVERSIDADE FEDERAL DE SANTA CATARINA

CURSO DE PÓS-GRADUAÇÃO EM CIÊNCIA DA COMPUTAÇÃO

IMPLEMENTANDO APLICAÇÕES DISTRIBUÍDAS UTILIZANDO CORBA E DCOM: UM ESTUDO DE CASO VOLTADO À ÁREA DE BANCO DE DADOS

Dissertação submetida à Universidade Federal de Santa Catarina como parte dos requisitos à obtenção do grau de Mestre em Ciência da Computação.

ANAMÉLIA CONTENTE DE SOUZA

Orientador: Prof. Dr. Vitório Bruno Mazzola

Florianópolis

Novembro de 1999

IMPLEMENTANDO APLICAÇÕES DISTRIBUÍDAS UTILIZANDO CORBA E DCOM: UM ESTUDO DE CASO VOLTADO À ÁREA DE BANCO DE DADOS

Anamélia Contente de Souza

'Esta dissertação foi julgada adequada para obtenção do Título de Mestre em Ciência da Computação, Área de Concentração *Sistemas de Computação*, e aprovada em sua forma final pelo Curso de Pós-Graduação em Ciência da Computação da Universidade Federal de Santa Catarina'.

Prof. Vitório Bruno Mazzola
Orientador

Prof. Fernando Álvaro Ostuni Gauthier
Coordenador do Curso de Pós-Graduação em Ciência da Computação

Banca Examinadora:

Prof. João Bosco da Mota Alves

Prof. Roberto Willrich

Prof. Vitório Bruno Mazzola

Para meus pais, com amor.

Agradecimentos

Ao Curso de Pós-Graduação em Ciência da Computação e a Universidade Federal de Santa Catarina pela infra-estrutura e organização que viabilizaram o desenvolvimento deste trabalho.

Aos professores João Bosco da Mota Alves, Roberto Willrich e Vitório Bruno Mazzola, por terem julgado este trabalho.

Ao Prof. João Bosco Mangueira Sobral pela atenção despendida em várias discussões sobre a Plataforma Corba.

Ao meu orientador, Prof. Vitório Bruno Mazzola, pela dedicação e pelos valiosos ensinamentos.

À minha amiga-irmã Patrícia Lima Seixas, pelo apoio e companheirismo.

Aos meus amigos João Neto, Claudio Blanco e Sérgio Barra, pelo apoio e acolhimento.

Agradeço especialmente aos meus pais, Aldo Avanir e Ana Leopoldina e meu irmão Éric Avanir, pelo incentivo, carinho e paciência.

RESUMO

O presente trabalho trata do estudo de plataformas para o desenvolvimento de aplicações distribuídas, em particular as arquiteturas CORBA e DCOM. O objetivo perseguido neste estudo é a realização de um aplicativo sobre ambas arquiteturas, visando observar os seus desenvolvimentos diante de uma situação real. Como forma de atingir este objetivo, as duas arquiteturas serão apresentadas com base na documentação disponível e os principais conceitos associados a estas serão apresentados e comparados, quando possível. A aplicação desenvolvida consiste do Banco de Dados de um hotel, o PitAnita's Hotel. Tal aplicação foi implementada através do ambiente Delphi 4.

ABSTRACT

This paper deals with the study of CORBA and DCOM architectures for the development of distributed applications. The objective pursued in this study is the realization of an application on both architectures to observe the development of each one in a real situation. As a way to reach this objective, both architectures will be presented based on the avaiable documentation and the main features related with them will be presented and compared, when possible. The developed application consists of a hotel's Database, the PitAnita's Hotel. Such application was implemented on the environment of Delphi4.

SUMÁRIO

LISTA DE FIGURAS

I CAPÍTULO
INTRODUÇÃO

O desenvolvimento de software segundo a tecnologia de Objetos tem se revelado uma abordagem com inúmeros benefícios. Dentre estes, pode-se destacar a modularidade, flexibilidade e reusabilidade, características essenciais para a qualidade e a produtividade do desenvolvimento de software.

Considerando a diversidade de plataformas computacionais e de linguagens de programação disponíveis atualmente, a concepção de uma aplicação deve levar em conta a possibilidade de utilização de componentes (ou Objetos) concebidos nas diferentes linguagens e capazes de executar em diferentes ambientes.

Visando proporcionar um ambiente de desenvolvimento capaz de manipular a heterogeneidade e a distribuição das aplicações atuais, o OMG (*Object Management Group*) definiu a arquitetura CORBA (*Common Object Request Broker Architecture*), uma arquitetura de suporte ao desenvolvimento voltada para aplicações distribuídas heterogêneas orientadas a Objetos. Desde o início dos anos 90, quando foram publicados os primeiros documentos relativos à arquitetura CORBA, um grande número de trabalhos foi iniciado no sentido de aprimorar a arquitetura e de apresentar implementações compatíveis com sua especificação.

Mais recentemente, a Microsoft apresentou a especificação de um ambiente com a mesma orientação do CORBA, a arquitetura DCOM, que tem sido vista como uma proposta concorrente àquela proposta pelo OMG.

Considerando as similaridades de objetivos, um estudo aprofundado das duas propostas se faz necessário, assim como o estabelecimento de parâmetros de comparação que permitam avaliar os pontos positivos e negativos de cada proposta. O encaminhamento de tal estudo permitirá avaliar as duas arquiteturas não no sentido de eliminar uma das duas, mas objetivando identificar em que situação a adoção de uma das arquiteturas é a boa opção. Outro aspecto importante a se realizar é a implementação de uma aplicação em ambas as arquiteturas, de forma a se obter uma visão prática do desenvolvimento de cada uma.

No segundo capítulo serão vistos alguns aspectos relacionados aos conceitos utilizados tanto pela arquitetura CORBA quanto DCOM, tais como os sistemas distribuídos, modelo cliente/servidor e orientação a objeto. Serão apresentadas também outras plataformas para ambientes distribuídos que surgiram antes destas duas, o DCE (*Distributed Computing Environment* – Ambiente para Computação Distribuída) e o ODP (*Open Distributed Processing* – Processamento Distribuído Aberto).

O terceiro capítulo é sobre a Plataforma CORBA. Neste será feito um estudo mais aprofundado sobre todo o funcionamento desta arquitetura, como ela surgiu, quais são os seus componentes, como eles interagem, quais as suas vantagens e os problemas ainda existentes, e onde ela pode ser executada.

A arquitetura DCOM é apresentada no quarto capítulo. O objetivo deste é basicamente o mesmo do terceiro, sendo que sobre o DCOM. Aqui será feito um levantamento sobre o seu histórico, seus principais conceitos e componentes, como ocorre a criação e interação destes, e onde esta arquitetura pode ser aplicada.

No quinto capítulo serão levantados alguns aspectos relevantes à adoção de plataformas para aplicações distribuídas orientadas a Objetos, onde será apresentado o comportamento de cada uma destas arquiteturas em relação aos mesmos. É um estudo comparativo teórico.

Já no sexto capítulo é realizado um estudo prático dessas arquiteturas. Uma aplicação de Banco de Dados Distribuídos foi desenvolvida sobre ambas arquiteturas, possibilitando um conhecimento real sobre algumas de suas características vistas nos capítulos anteriores.

O resultado e as dificuldades enfrentadas para a realização do capítulo anterior, são relatados neste capítulo que é o de número sete. Este também apresenta algumas propostas para trabalhos futuros.

II CAPÍTULO
APLICAÇÃO DISTRIBUÍDA ORIENTADA A OBJETOS: CONCEITOS E PLATAFORMAS

2.1. SISTEMAS DISTRIBUÍDOS

Nos primórdios da computação, os computadores apresentavam por um custo extremamente elevado e o processamento das informações era feito de forma independente (*stand-alone)*, o que limitava o aproveitamento do potencial destes equipamentos em muitas áreas de aplicação.

Entretanto, a evolução registrada na área da microeletrônica propiciou contínuas reduções de custo dos computadores, oferecendo a possibilidade de multiplicação da quantidade de computadores numa organização. O aumento das capacidades de processamento (em termos de velocidade e de sofisticação no conjunto de instruções dos processadores) e de armazenamento que tem ocorrido nos últimos anos é outro aspecto a ser considerado. Além disso, o advento dos sistemas de comunicação entre computadores, com arquiteturas e protocolos de rede cada vez mais sofisticados contribuem para a difusão do conceito de sistemas computacionais distribuídos, em contraste aos sistemas centralizados que possuem um único conjunto dotado de processador, memória, periféricos e terminais [11].

O problema existente nos sistemas distribuídos, é que ele requer softwares radicalmente diversos daqueles que rodam nos sistemas centralizados. Aplicações como controle de tráfego aéreo, operações bancárias, vídeoconferência, são alguns exemplos de aplicações que necessitam ser realmente distribuídas.

Uma aplicação é considerada distribuída se ela possuir pelo menos dois componentes sendo executados em máquinas diferentes. Contudo, devido tais aplicações não serem projetadas para ser distribuídas, sua escalabilidade fica limitada, assim como sua capacidade de desenvolvimento.

Um grande número de aplicações distribuídas opera segundo uma política de Cliente-Servidor, onde o controle da forma como os usuários irão comunicar-se e cooperar é um aspecto fundamental. Caso estas aplicações fossem desenvolvidas para ser distribuídas e também executando os componentes adequados nos devidos lugares, isto beneficiaria tanto o usuário quanto a rede, que seria otimizada e também os recursos do computador.

Aplicações distribuídas permitem ao gerente do sistema, maior flexibilidade no desenvolvimento e controle do mesmo. O problema da escalabilidade, por exemplo, pode ser

resolvido através da criação de componentes em máquinas de menor custo, a medida que a carga de trabalho aumenta. Um simples servidor fica encarregado de iniciar todos os componentes, diminuindo assim o total dos custos [30].

O que se verifica também, é que há uma necessidade de se reaproveitar programas, códigos ou sistemas, já existentes. Isto é o que se chama de sistemas herdados.

Sistemas heterogêneos, são aqueles que englobam diversos tipos de sistemas operacionais como, mainframes, estações, servidores UNIX, etc. Os tipos de protocolos e redes, também podem ser os mais diversos possíveis: ATM, Ethernet, FDDI, etc.

Apesar de haver uma grande dificuldade para se administrar um sistema heterogêneo, ele possui uma grande vantagem, que é reunir o melhor dos hardwares e softwares de diversos fabricantes [11].

E é para ajudar nesta administração que entra a Orientação a Objeto. É por esta razão que a boa modelagem do objeto é fundamental. Um objeto bem modelado, poderá ser reaproveitado em outras aplicações. Também, quando for necessária alguma atualização, esta poderá ser feita em um único objeto, ou em um grupo destes. A detecção e correção de erros também se tornará mais fácil, uma vez que estamos lidando com objetos que representam elementos do mundo [13].

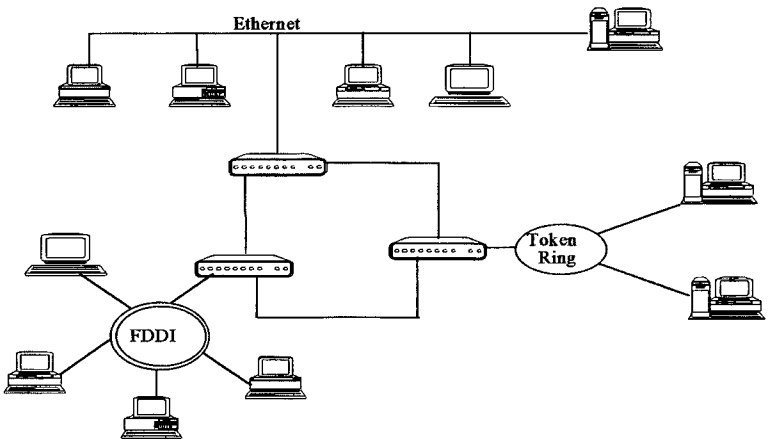

Figura 2.1- Exemplo de um Sistema Distribuído Heterogêneo [13]

2.2. MODELO CLIENTE/SERVIDOR

O modelo cliente/servidor se propõe a estabelecer a comunicação entre vários processos, estejam eles na mesma máquina, ou não, sem a necessidade de se estabelecer uma conexão prévia. Este modelo é baseado em um protocolo bem simples do tipo solicitação/resposta, o que faz com que uma de suas vantagens seja justamente a simplicidade.

Ele funciona da seguinte forma: um processo será o **cliente**, que é aquele que irá solicitar algum serviço de um outro processo, e o outro será o **servidor,** que é quem irá realizar o serviço solicitado. O processo que roda como cliente, também pode rodar como servidor e vice-versa. O cliente irá mandar através do seu *stub* a mensagem. Este irá empacotar a mesma e enviá-la ao *kernel* do cliente, que se encarregará de encontrar o *kernel* do servidor. Este, por sua vez, irá encontrar o *stub* do servidor, que desempacotará a mensagem e irá enviá-la ao processo servidor. Quando este terminar de executar a mensagem, ele retorna a resposta ao seu *stub*, que irá fazer todo o processo de volta.

A própria resposta enviada pelo servidor já serve como uma confirmação do recebimento da mensagem. Isto faz com que possa ser utilizada uma pilha de protocolos bem menor do que no modelo em camadas, o que torna também este modelo mais eficiente.

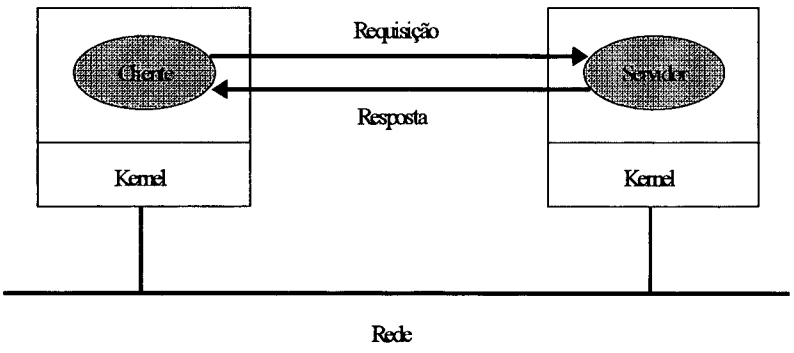

Figura 2.2 – Modelo Cliente/Servidor [11]

2.3. APLICAÇÕES ORIENTADAS A OBJETO (O.O.)

As técnicas convencionais utilizadas para a construção de softwares, têm tido problemas no sentido de que não conseguem acompanhar a evolução do hardware, além de não incorporar de forma satisfatória novos recursos como multimídia, interfaces gráficas, computação cliente-servidor e ambientes distribuídos.

O objetivo da orientação a objeto, é justamente o de amenizar tal situação, através de suas características básicas e essenciais que são o encapsulamento, heranças nas classes e polimorfismo nas operações [13].

Nos parágrafos a seguir, são apresentados os principais conceitos que regem o desenvolvimento orientado a objetos:

* Encapsulamento: é o conceito que garante a transparência quanto à localização, isto é, a implementação dos objetos não precisarão ser modificadas, caso eles mudem de lugar. Encapsulamento significa que só estará disponível para visualização por parte de usuários comuns uma "máscara", ou seja, o "conteúdo real", não é visto por estes usuários. No caso da orientação a objeto, isto funciona da seguinte forma, o cliente só verá a interface (máscara) do objeto e nunca a sua implementação (conteúdo real). Por isso é que alterações podem ser realizadas no objeto, sem que o cliente tome conhecimento.

* Herança: é a propriedade de objetos novos serem gerados a partir de outros já existentes, de acordo com as características destes, herdando-as. Com isto, é possível economizar um grande trabalho que seria gasto na implementação e configuração destes objetos.

* Polimorfismo: é a capacidade de operações que são utilizadas por um determinado conjunto de objetos, gerarem resultados diversos. Por exemplo, a operação volume, pode ser aplicada tanto a um cubo, quanto a um cone, gerando resultados diferentes.

Além dessas características, a orientação a objeto também utiliza alguns conceitos importantes. Abaixo , estão descritos alguns deles:

* Sistemas de Objetos: é um conjunto de objetos que separa os clientes dos objetos prestadores de serviços, através de uma , interface.

* Requisições: são os pedidos feitos a um objeto.

* Operação: é o serviço que o objeto realiza.

* Parâmetros: são dados, que podem ser passados junto com a requisição, que fornece informação da mesma. Podem haver parâmetros também na resposta da requisição.

* Exceções: quando ocorre algum problema, são retornadas mensagens ou são ativados métodos de tratamento de exceções que estão associadas as requisições.

* Contextos: estes também estão associados às requisições e são retornados no caso de haver informações capazes de alterar a performance de uma requisição.

* Invocação: significa especificar o objeto requisitado, junto com a operação a ser realizada e os parâmetros de entrada e saída da mesma.

* Interface ou Descrição de Interface: define quais serviços podem ser executados por um dado objeto.

* Atributos: os dados encapsulados pelo objeto, são atribuídos à interface, podendo ter seu valor lido ou alterado através de um par de operações.

A orientação a objeto, como o próprio nome diz, é centrada em objetos. Tais objetos são componentes de software, que modelam elementos do mundo real. Isto significa dizer que tais elementos (objetos), possuem dados e são capazes de manipular tais dados.

É importante que o objeto seja bem modelado, pois uma das vantagens da orientação a objeto consiste na reutilização destes objetos em outras aplicações, ou quando uma atualização for realizada, a alteração poderá ser feita somente em um único objeto ou em um grupo deles.

Os objetos são criados de acordo com uma determinada classe, a qual chamamos de "*template*", que terá total conhecimento de suas propriedades. É o *template* que dirá quais operações este objeto poderá efetuar, ou seja, quais mensagens poderá receber.

O desenvolvimento de softwares baseados na orientação a objetos, consolidou-se bastante desde o aparecimento das metodologias que seguiram esta abordagem. A construção de aplicações orientadas a objeto, busca aumentar a reusabilidade, a portabilidade, a interoperabilidade nos diversos ambientes heterogêneos, de forma que esta heterogeneidade fique transparente ao programador. Acontece que, devido a concorrência existente entre as diversas empresas envolvidas, não havia um padrão definido, impossibilitando assim, a construção de um ambiente distribuído orientado a objeto, que englobe várias máquinas, com diferentes sistemas operacionais, comunicando-se remotamente.

O O.M.G. (*Object Management Group*), é uma instituição encarregada da definição de padrões, procurando resolver problemas relacionados ao desenvolvimento de software distribuído como a compatibilidade entre especificações. Ele define tais padrões, através de Requisições por Informações (*Request For Information* - RFI) e Requisições por Propostas (*Requests For Proposals* - RFP), enviadas aos seus membros que, realizam a análise necessária e chegam a uma conclusão única.

Foi ele que elaborou um modelo onde, um objeto alocado em uma determinada máquina, poderia acessar um serviço disponível em uma outra máquina, situada remotamente, através de uma interface intermediária denominada ORB (*Object Request Broker*).

O ORB está situado entre o objeto, que é a camada de aplicação no modelo OSI, e o sistema operacional. Ele possui diversas características, que o permite estabelecer esta comunicação através da rede.

2.4. PLATAFORMAS PARA AMBIENTES DISTRIBUÍDOS HETEROGÊNEOS

2.4.1. OPEN DISTRIBUTED PROCESSING – ODP (PROCESSAMENTO DISTRIBUÍDO ABERTO)

Seja qual for o sistema que se pretende desenvolver, é necessário analisar antes de tudo, como deve ser armazenada e tratada a informação. Sem informação não há sistema e a má localização e segurança desta, pode causar sérios danos ao mesmo.

A crescente implantação de redes locais, metropolitanas e de longa distância, fez com que aumentasse também o processamento distribuído, o qual tem o objetivo de:

- Prevenir falhas: a replicação dos dados fará com que no caso de falha em um computador, as informações ali contidas não sejam perdidas e nem tão pouco fiquem inacessíveis, pois as mesmas estão armazenadas em outro(s) computador (es);
- Permitir a diminuição do tempo de acesso: existirão vários pontos com os mesmos dados, o que permitirá ao sistema buscar a informação no ponto mais próximo;
- Possibilitar o processamento em paralelo, o que permite uma solução eficaz para grandes problemas;
- Copiar para diferentes grupos de acordo com o gerenciamento da empresa.

A conclusão que se tem de acordo com os itens citados acima, é que a distribuição do sistema de informação é inevitável e necessária. Portanto, é preciso que haja uma padronização deste processamento para que o mesmo seja consistente e confiável.

Outro fator que leva a padronização são os equipamentos que compõem uma rede, sejam eles de hardware ou software. Tais equipamentos possuem uma grande heterogeneidade, o que faz com que aumente o problema de manter a integridade e segurança das informações do processamento distribuído aberto.

Foi para atingir este objetivo que a ISO (*International Standard Organization* – Organização de Padrão Internacional), IEC (*International Electrotechnical Commission* – Comissão Eletrotécnica Internacional) e ITU (*International Telecommunications Union* – União de Telecomunicações Internacional), se reuniram e criaram em 1987, o padrão

internacional do **Modelo de Referência do Processamento Distribuído Aberto (RM-ODP)** [32].

Este modelo cria uma arquitetura na qual o suporte para distribuição, interação, interoperabilidade e portabilidade, podem ser integrados [25].

"O objetivo principal dos padrões do ODP, é possibilitar a interação de aplicações e o compartilhamento de dados através das redes de computadores indo além dos limites organizacionais e nacionais" [30].

O RM-ODP tem os seguintes objetivos:

- Portabilidade de aplicações através das plataformas heterogêneas;
- Interação entre sistemas ODP: trocas de informações e o uso adequado de funcionalidades através do sistema distribuído;
- Transparência de distribuição: omitir as conseqüências da distribuição tanto dos programadores quanto dos usuários.

O modelo de referência não padroniza os componentes do sistema e nem influencia na escolha da tecnologia. Ele simplesmente fornece uma forma para que as partes de um sistema ODP sejam organizados coerentemente.

2.4.2. DISTRIBUTED COMPUTING ENVIRONMENT – DCE (AMBIENTE DE COMPUTAÇÃO DISTRIBUÍDO)

O DCE é uma arquitetura que foi desenvolvida pela OSF (*Open Software Foundation*, atualmente chamada de *Open Group*), com o objetivo de integrar aplicações de ambientes distribuídos heterogêneos. Ele foi desenvolvido bem antes do paradigma da programação orientada a Objetos, sendo portanto, baseado na programação procedimental [2] [7] [32] [17].

Segundo [7], o DCE é talvez mais que um sistema de informação ou banco de dados distribuídos. Ele é um sistema operacional distribuído, embora não substitua um sistema operacional existente. Ao invés disso, ele é ajustado no topo deste. O mais correto seria vê-lo como uma camada intermediária entre um sistema operacional convencional e as aplicações distribuídas, que pretendem ser executadas no mesmo. A figura 2.4 ilustra os diferentes componentes de uma aplicação desenvolvida em DCE, cuja funcionalidade é descrita nos parágrafos que seguem:

Figura 2.3 – Arquitetura DCE [2] [7] [17]

- Subprocessos (*Threads*): é responsável pela criação, sincronização e gerenciamento de múltiplas subprocessos de controle dentro de um único processo. Será necessário somente no caso de o sistema operacional que está sendo utilizado não possuir tal capacidade.
- Chamada Remota a Procedimento (RPC): é o mecanismo pelo qual uma mensagem que está no formato do programa cliente, é convertida para o formato da rede, de forma que esta possa trafegar pela mesma e ser entregue ao servidor destino e vice-versa. A transparência é implementada através da utilização de outros serviços do DCE, tais como o serviço de diretório.
- Serviço de Tempo Distribuído (DTS): sincroniza o *clock* de todos os nós da rede com o *clock* que controla a rede como um todo.
- Serviço de Segurança: é responsável pela autenticação, comunicação segura e autorização.
- Serviço de Diretório: suporta a administração local dos domínios do DCE denominados de **células** e **resolução de nomes inter–células**. Este serviço consiste de um **Serviço de Diretório de Célula** (CDS), **Serviço de Diretório Global** (GDS), **Serviço de Nome de Domínio** (DNS, não fornecido, mas utilizado pelo DCE) e um **Agente de Diretório Global** (GDA).

- Sistema de Arquivo Distribuído (DFS): é uma aplicação do DCE que facilita o acesso a arquivos em servidores de arquivos remoto, fornece transparência de localização e garante uma alta performance.

O DCE não é apenas um pacote de software que pode ser instalado em um servidor. Na realidade, os seus componentes estão localizados entre as aplicações e os serviços de rede, tanto no cliente quanto no servidor, uma vez que ele foi construído segundo o paradigma do modelo cliente/servidor.

Embora ele seja uma arquitetura de multicamadas que contém vários serviços básicos, o seu modelo cliente/servidor oculta detalhes destes serviços dos usuários finais. Na verdade, os componentes do DCE representaram uma parte integral do modelo de computação distribuída que foi desenvolvido por grupos de padronizações, tais como a ISO (*International Standards Organization* – Organização de Padrões Internacionais) [2].

2.4.3. COMMON OBJECT REQUEST BROKER ARCHITECTURE – CORBA (ARQUITETURA INTERMEDIÁRIA PARA SOLICITAÇÃO DE OBJETOS COMUNS)

O CORBA é uma plataforma para ambientes distribuídos orientados a Objetos, que surgiu a partir da necessidade de se criar um padrão para a interação dos objetos dentro das aplicações.

Esta necessidade ocorreu devido o OMG (*Object Management Group* – Grupo de Gerenciamento de Objeto), em 1989-1990, ter desenvolvido uma Arquitetura de Gerenciamento de Objetos (*Object Management Architecture* – OMA) que descrevia a maneira que as aplicações deveriam interoperar.

Esta arquitetura possui quatro elementos que integrados formam a base de toda a estrutura do CORBA. Dentre eles, o ORB é o componente mais importante, pois é através dele que é estabelecida a comunicação entre os objetos.

O CORBA foi criado pelo OMG para dar suporte à criação e utilização dos componentes de software na tecnologia de objetos. Os objetos deste são componentes binários que clientes remotos podem acessar através de invocações de métodos. A linguagem e o compilador utilizados para criar os objetos servidores são totalmente transparentes aos clientes. Estes não necessitam saber onde os objetos distribuídos estão localizados e nem o sistema operacional em que eles são executados. Este é o fator da transparência, essencial para a interoperação de objetos em ambientes heterogêneos.

A única informação necessária é da interface do objeto servidor. Esta funciona como um contrato entre clientes e servidores. É através dela que os clientes tomam conhecimento sobre os serviços oferecidos por cada objeto, os parâmetros necessários para os mesmos, as classes da qual este objeto deriva, seus atributos, os tipos de eventos e as exceções emitidas.

O CORBA é uma tecnologia aberta, isto é, ele não pertence a nenhum grupo específico e sim a um grupo de mais de 700 empresas que se uniram para criar um padrão para a integração de objetos distribuídos em ambientes heterogêneos. Isto possibilita que o ele seja executado na maioria das grandes plataformas existentes, inclusive UNIX, Windows NT e Windows 95.

Desde o início dos anos 90, quando foram publicados os primeiros documentos relativos à arquitetura CORBA, um grande número de trabalhos foi iniciado no sentido de aprimorar a arquitetura e de apresentar implementações compatíveis com sua especificação. Um estudo mais aprofundado e detalhado sobre esta arquitetura, será apresentado no capítulo 3, de forma que se possa obter uma visão da sua aplicabilidade, através do conhecimento de seu funcionamento e de algumas de suas vantagens e desvantagens.

2.4.4. Distributed Component Object Model – DCOM (Modelo de Componente de Objeto Distribuído)

O COM (*Component Object Model*) é uma especificação e implementação desenvolvida pela Microsoft, que fornece uma estrutura para a integração de componentes.

Para construir os sistemas, os desenvolvedores podem utilizar componentes já existentes, de diferentes vendedores, que se comunicarão através do COM. Isto faz com que a estrutura suporte a interoperabilidade e reusabilidade de objetos distribuídos.

O DCOM surgiu em 1996 como uma extensão do COM. Ele permite a interação de componentes através das redes, enquanto que o COM só suportava esta interação entre objetos com espaços de endereços diferentes, porém na mesma máquina.

Tanto para a criação de componentes, quanto para a interação dos mesmos, o DCOM define uma API (*Application Programming Interface* – Interface de Programa de Aplicação). Contudo, para que eles possam interagir, estes componentes necessitam aderir a uma estrutura binária especificada pela Microsoft. Feito isso, poderá ocorrer a interoperabilidade de componentes escritos nas mais diversas linguagens.

O elemento intermediário na comunicação entre os clientes e os objetos COM, é a interface. Esta fornece um conjunto de métodos relacionados, através de uma estrutura binária, que descreve os serviços suportados pelos objetos.

Cada objeto pode possuir várias interfaces, cada uma descrevendo um serviço específico. Estas interfaces são especificadas por uma Linguagem de Definição de Interface (IDL) da Microsoft, que é uma extensão da IDL utilizada pelo DCE.

O DCOM foi construído no topo do DCE. Ele adaptou a estrutura de comunicação entre ambientes heterogêneos do DCE na sua estrutura, sendo que para objetos distribuídos.

Um estudo mais específico sobre a plataforma DCOM será feito no capítulo 4. Através deste estudo pretende-se obter um embasamento para o capítulo 5, onde será feita uma comparação entre esta plataforma e a plataforma CORBA.

2.5. Conclusões

Neste capítulo, foram apresentados os conceitos básicos relativos ao tema tratado neste documento, uma breve apresentação de aspectos como os modelos Cliente-Servidor e o paradigma de Orientação a Objetos. Foram apresentados também os fatores de destaque das diversas iniciativas voltadas à definição de plataformas de suporte ao desenvolvimento de aplicações distribuídas, dentre as quais destacam-se CORBA e DCOM, como soluções promissoras a este problema.

Perseguindo o objetivo maior do trabalho, será apresentado, nos capítulos que seguem os resultados dos estudos realizados sobre estas duas plataformas, procurando levantar pontos comuns e diferenças existentes para poder estabelecer parâmetros de comparação entre as propostas.

III CAPÍTULO
CORBA

3.1. ARQUITETURA CORBA

O CORBA é uma arquitetura que foi adotada pela OMG e tem sua base na computação Cliente/Servidor e no paradigma de Orientação a Objetos.

O CORBA define as aplicações distribuídas como conjuntos de objetos Clientes e Servidores, que interagem e cooperam para a realização das tarefas relacionadas a estas aplicações. Dentro desta política, objetos Clientes vão requisitar serviços aos objetos Servidores, os quais deverão executá-los e encaminhar respostas relativas aos resultados da realização dos serviços solicitados. O encaminhamento das solicitações e das respostas aos serviços é gerenciado por um componente da plataforma, o ORB (*Object Request Broker*) que se encarrega da manipulação da heterogeneidade inerente ao sistema. Para que a aplicação possa funcionar sem problemas, os objetos Clientes deverão manter o conhecimento dos serviços realizados pelos diferentes objetos do sistema, de modo a que possa encaminhar corretamente suas solicitações [13] [1].

Os objetos também podem exercer a função de clientes, isto é, eles podem solicitar a um outro objeto, um determinado serviço, que seja necessário para que ele possa continuar efetuando o seu. O cliente pode obter informações dos serviços do objeto, nas interfaces dos mesmos.

Estas interfaces tanto podem fornecer estas informações ao cliente, quanto à infra-estrutura de comunicação. O formato das mensagens recebidas e enviadas pelos objetos, não fica visível, de forma que seja assegurada a transparência na comunicação entre o cliente e o objeto.

Estes objetos recebem um identificador único, para diferenciá-los um do outro, independente do local onde se encontrem, chamada "referência do objeto".

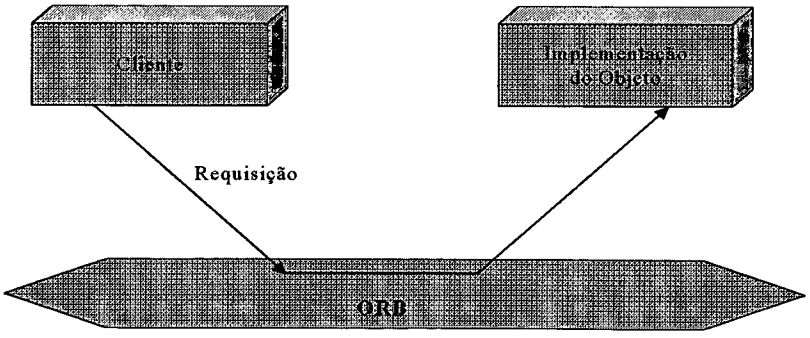

Figura 3.1 - Requisição enviada através do ORB [13]

3.2. ARQUITETURA DE GERENCIAMENTO DE OBJETOS - OMA (*OBJECT MANAGEMENT ARCHITECTURE*)

A OMA, foi criada pelo OMG, com o objetivo de integrar as aplicações baseadas em objetos distribuídos.

Sua estrutura é baseada basicamente em dois elementos: o ***Modelo de Objeto***, que é quem define o objeto que será distribuído pelo sistema heterogêneo e o ***Modelo de Referência***, que é quem define as características da integração destes objetos.

O RFP (*Request for Proposal* – Propostas para Requisições) é utilizado para que os Modelos de Objetos e de Referência, possam ser compatíveis com especificações anteriormente utilizadas e, com isso tornar possível a construção de sistemas de objetos distribuídos interoperáveis em sistemas heterogêneos.

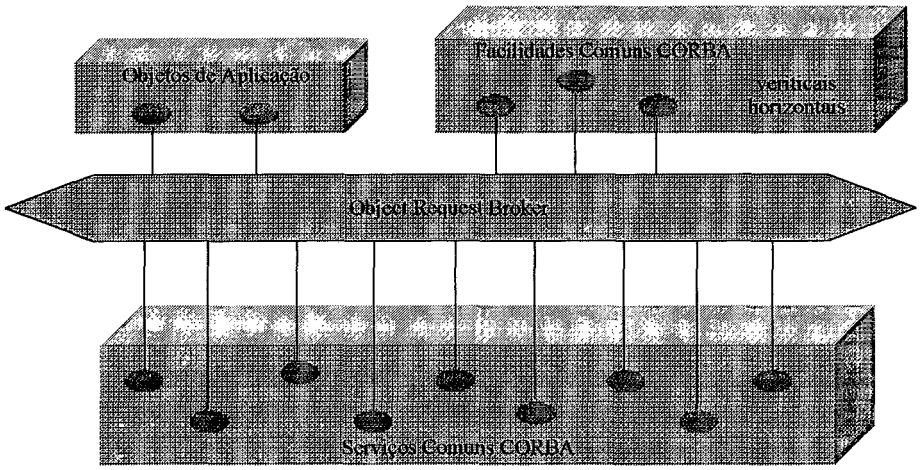

Figura 3.2 - Modelo de Referência OMA [1]

O CORBA possui toda sua estrutura baseada na arquitetura OMA. Esta define a comunicação entre os objetos distribuídos através de quatro elementos:

1. ***Objetos de Aplicação***: são os objetos criados pelo usuário, que possuem características definidas de acordo com os seus objetivos. Eles também são tidos como os usuários finais de toda a infraestrutura CORBA.
2. ***Facilidades Comuns CORBA***: são componentes definidos em IDL que fornecem serviços para o uso direto das aplicações de objetos. Estes estão divididos em duas categorias que são, horizontal e vertical. Eles definem regras de integração para que os componentes possam colaborar efetivamente. Existem quatro tipos de facilidades horizontais:

- *User Interface* (Interface de Usuário) – são serviços de edição similares aos fornecidos pelo OpenDoc e OLE;
- *Information Management Services* (Serviços de Gerenciamento de Informação)– inclui o armazenamento de documentos compostos e facilidades de troca de dados;
- *Systems Management Services* (Serviços de Gerenciamento de Sistemas)- definem interfaces para o gerenciamento, configuração, instalação, operação e recuperação de componentes de objetos distribuídos;
- *Task Management Services* (Serviços de Gerenciamento de Tarefas) – inclui fluxo de trabalho, longas transações, agentes e correio eletrônico.

As facilidades verticais fornecem interfaces definidas em IDL para segmentos do mercado tais como, saúde, varejo e finança.

3. ***Serviços Comuns CORBA***: são serviços (interfaces e objetos), que possibilitam a implementação e utilização de objetos. Eles definem uma estrutura de objetos de baixo nível, que ampliam as funcionalidades do ORB, sendo utilizados para criar um componente, nomeá-lo e introduzí-lo no ambiente.
4. ***ORB***: é o elemento que permite que objetos emitam requisições em um ambiente distribuído e heterogêneo, de forma transparente.

3.3. ORB (*OBJECT REQUEST BROKER*)

Clientes e objetos que se comunicam no CORBA, utilizam diferentes linguagens de programação, ordenação de bytes interna da máquina, localização e outros aspectos que compõe um sistema heterogêneo distribuído. Portanto, é necessário que a interface com a qual o cliente tem contato direto, e esta com o objeto, seja independente de tais fatores. Esta interface é denominada de ORB (*Object Request Broker*).

O ORB, além de garantir, a transparência dos fatores acima citados, ele também é responsável por ativar um objeto se necessário, ao qual uma requisição se destina, de forma transparente, não importando se o objeto está localizado local ou remotamente [1].

Como já foi mencionado, um cliente precisa informar a que objeto se destina sua requisição. Isto é feito através da referência do objeto. Existem diversas maneiras de um cliente obter as referências sobre outros objetos do sistema. Uma delas é no momento da criação do objeto. No CORBA, este processo ocorre como se fosse uma requisição como outra qualquer, feita pelo cliente, ao qual retorna a referência a objeto.

Outra forma é através do serviço de procura que o cliente solicita. Este pode obter referência a objetos por nome ou por propriedades através do "serviço de nomes" e do "serviço de *trading*".

A terceira maneira é solicitar ao ORB, que a referência seja convertida em *string* e armazenada em um arquivo ou Banco de Dados. Esta conversão pode ser desfeita.

As referências iniciais para os clientes é obtida através de um serviço de nomes, mantido pelo ORB, o qual fornece referência a objetos às aplicações dos serviços de diretórios mais utilizados.

Agora, se processo de criação de objetos é feito pelos clientes, em cima de objetos já existentes, como é que é obtida a referência inicial? Da seguinte forma: o ORB mantém um serviço de nomes que fornece referência a objetos às aplicações dos serviços de diretórios mais utilizados.

Isto faz com que a estrutura do ORB permaneça bem simples, que é uma das características básicas do CORBA, se preocupando somente com a comunicação e a infra-estrutura de ativação para as aplicações de objetos distribuídos, que é o que ORB deve realmente fazer. Deixando para os demais componentes do OMA, o máximo de funcionalidade possível [13].

3.4. LINGUAGEM DE DEFINIÇÃO DE INTERFACE - IDL (*INTERFACE DEFINITION LANGUAGE*)

Para que um cliente emita uma requisição a um objeto, ele necessita saber quais serviços tal objeto pode executar. Tais informações são fornecidas pela interface do objeto.

Uma vez que o CORBA tem por objetivo interoperar com sistemas heterogêneos, nos quais nem todas as linguagens de programação são suportadas em todas as plataformas disponíveis, é importante para ele lidar com a IDL, pois esta é uma linguagem de definição e não de implementação [3].

A OMG IDL possui interfaces similares às classes do C++ e interfaces Java, com algumas alterações feitas pelo ANSI devido a padronização e o acréscimo de palavras-chave para suportar tal arquitetura definida pelo OMG.

```
//OMG IDL
interface Exemplo1
{
long op1 (in long arg1);
};
```

Figura 3.3 - Exemplo de Definição de uma Interface em IDL [13]

Esta definição especifica uma interface nomeada **Exemplo 1** que suporta uma operação denominada **op1**. A operação **op1** necessita do parâmetro de entrada **arg1** e retorna uma referência do tipo **long**.

Assim como existem os sistemas herdados, também existem as interfaces herdadas.

As interfaces originais, são chamadas de interfaces-base e as interfaces herdadas, de interfaces derivadas.

As interfaces derivadas herdam todas as operações das interfaces-base. Portanto, as referências de objetos das interfaces derivadas podem substituir, em qualquer local, as referências de objeto das interfaces-base.

```
//o mesmo exemplo visto anteriormente
Interface Exemplo1
{
long op1 (in long arg1);
};

//exemplo da herança
#include "exemplo1.dll"
interface exemplo2:exemplo1    //herança
{
    void op2( );
}
```

Figura 3.4 - Exemplo de Herança em IDL [13]

Existe um caso especial de herança no CORBA, onde todas as interfaces são derivadas da interface **Objeto** definida no módulo CORBA. Agora, devido o fato desta herança ser padrão, não há a necessidade da declaração de herança nos objetos CORBA.

A figura abaixo mostra como seria esta declaração.

```
//CORBA::Object é a Interface-base para todas as interfaces
interface Exemplo1 : Object{...};
```

Figura 3.5 - Interface Object do CORBA [13]

Pode ocorrer também o caso de **Herança Múltipla**, onde uma interface derivada possui várias interfaces-base, sendo que a ordem em que estas são declaradas não importa.

Esta relação de herança e a simplicidade da OMG IDL, são muito importantes para o CORBA, pois garante sua utilização com um número maior de linguagens de programação, do COBOL ao Java.

3.5. MAPEAMENTO DE LINGUAGENS

A linguagem IDL, como já foi vista, é uma linguagem puramente declarativa, que define os limites dos componentes, assim como as suas interfaces. Portanto, ela não fornece as características devidas para a implementação de aplicações distribuídas.

Para isto, é feito o mapeamento desta linguagem para outras linguagens de programação como, C, C++, Samalltalk, ADA, COBOL e Java.

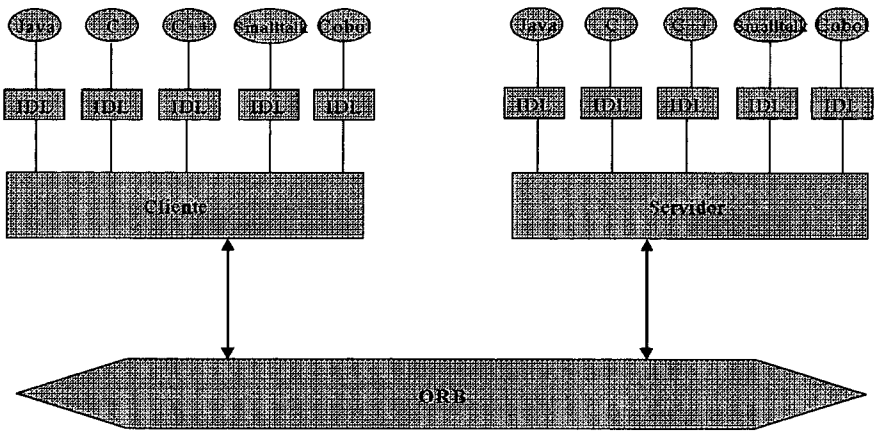

Figura 3.6 – Mapeamento da Linguagem IDL em outras Linguagens [1]

Este mapeamento deverá, antes de tudo, estar apto a expressar todas as construções utilizadas em IDL (tipos de dados, constantes, objetos, invocações de operações e outros). Além de ter cada tipo de dado IDL mapeado para a linguagem em questão, garantindo assim a interoperabilidade, deverá também haver um tipo de dado para representar a referência a objeto.

Um outro fator no mapeamento, é quanto ao ORB e aos Pseudo-objetos. Estes últimos não são objetos propriamente ditos do CORBA, mas sim interfaces ORB, que também não é um objeto do CORBA. Acontece que a especificação destas interfaces, agem como objetos normais, permitindo assim que aplicações manipulem o ORB, assim como manipulam qualquer outro objeto.

No caso de linguagem O.O., os objetos CORBA, são mapeados como objetos de linguagem de programação. Já em C, eles são mapeados como tipos abstratos de dados.

3.6. Repositório de Interfaces - IR (*Interface Repository*)

O ORB é o elemento através do qual clientes e implementações de objetos acessam vários serviços, não importando o adaptador de objeto utilizado. Para que tais serviços sejam acessados, eles precisam estar disponíveis e possuir uma interface comum, seja qual for a implementação do ORB utilizada.

Esta interface é o IR. É através dele que o ORB possibilita o acesso distribuído às implementações de objetos e disponibiliza aos elementos da arquitetura OMA as interfaces dos objetos especificados na linguagem IDL.

Para que o ORB possa interagir com objetos, ele precisa obter informações sobre estes e ter um local para armazená-las. Utilizando o IR, ele pode no momento de uma requisição (na invocação dinâmica que veremos posteriormente), verificar os tipos de parâmetros, as relações de herança entre interfaces e ajudar na interação de diferentes ORBs.

Clientes e implementações de objetos também podem ter acesso ao IR. Eles também podem gerenciar a instalação e distribuição de interfaces, quanto fornecer informações a utilitários [1].

3.6.1. Organização do Repositório de Interface

As interfaces são agrupadas em módulos, de maneira a facilitar o processo de identificação.

A localização da interface, pode ser obtida a partir de uma definição de interface por meio da interface do ORB e aí através de uma busca de uma seqüência de nomes pelo IR, ou localizando a definição de interface correspondente a um determinado identificador.

Na tabela abaixo, estão algumas das operações que possibilitam o acesso aos elementos do IR.

Operação	Descrição
lookup_id ( )	procura o objeto no depósito de interfaces que possua o identificador passado como parâmetro da chamada
contents ( )	usada para listar o conteúdo (constantes, definições de tipo, exceções, interfaces e módulos) do depósito de interfaces
lookup_name ( )	procura elementos do depósito de interfaces que possuam o nome especificado; pode ser especificado o tipo de elemento que se deseja
describe_contents ( )	usado para obter a descrição completa do conteúdo do depósito de interfaces, retornando o identificador e o nome de cada objeto contido

Tabela 3.1 - Operações de Acesso ao Depósito de Interfaces [13]

3.7. *STUBS* E *SKELETONS*

Os *stubs* e os *skeletons* realizam funções correspondentes, sendo que o *stub* é do lado do cliente e o *skeleton*, do objeto. São eles que implementam a arquitetura cliente/servidor no CORBA.

Tanto o *stub* quanto o *skeleton* são criados baseados na compilação da interface IDL do cliente e do objeto, respectivamente.

Quando um cliente emite uma requisição, ele só precisa dizer ao *stub* para qual objeto destina-se a requisição, não devendo importar se o objeto está local ou não. Isto é o que garante a transparência nos sistemas distribuídos. O *stub* localizará o objeto-destino e fará junto com o ORB o empacotamento da mensagem, de forma que seja possível enviá-la pela rede [3] [13].

Devido a esta capacidade de localizar objetos-destinos, os *stubs* também são denominados de *proxies* ou procuradores.

Os *skeletons* têm função parecida, com a diferença que eles irão receber requisições do ORB. A partir de então, deverão desempacotar a mensagem e entregá-la à implementação do objeto. Caso haja resposta, eles farão o processo inverso.

Uma vez que as mensagens são construídas na aplicação cliente e na implementação do objeto, *stubs* e *skeletons*, precisam ter conhecimento completo das interfaces OMG IDL dos objetos CORBA que são invocados. Isto é o que se chama de **Invocação Estática**.

O processo então é o seguinte:

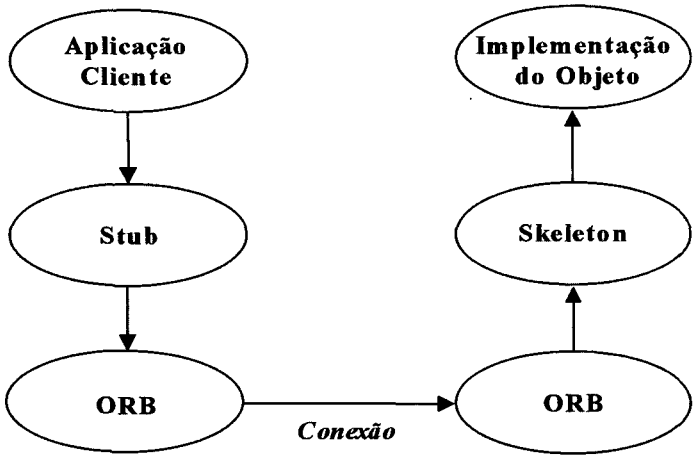

Figura 3.7 – Invocação Estática [1]

O *stub* deve transformar da linguagem de programação que é passada da aplicação do cliente, para uma forma que possa ser trafegada pela rede. E o *skeleton* faz o inverso. Ele transforma o que recebe para a linguagem de programação da implementação do objeto.

3.8. Invocação e Envio Dinâmico

A invocação e envio dinâmico surgiu para solucionar o problema da inflexibilidade da invocação estática, que não permitia que objetos criados após a compilação do *stub* ou *skeleton*, fossem acessados.

Na invocação estática, para que novos objetos pudessem ser acessados, era necessário parar o sistema ou realizar nova compilação. Isto já não e mais necessário na invocação dinâmica.

As duas interfaces suportadas pelo CORBA para a invocação dinâmica são o DII e o DSI.

O DII (Interface de Invocação Dinâmica), corresponde ao *stub* e o DSI (Interface *Skeleton* Dinâmica), ao *skeleton*.

Aqui, estas interfaces, diferentemente da invocação estática, são fornecidas diretamente pelo ORB, portanto não há necessidade de cada cliente ou objeto possuir a sua.

A vantagem da invocação dinâmica, é a transparência, pois a implementação nunca sabe se a sua requisição está sendo feito a um *stub* estático ou dinâmico [1].

3.8.1. INTERFACE DE INVOCAÇÃO DINÂMICA - DII (DINAMIC INVOCATION *INTERFACE)*

Na DII, clientes podem emitir requisições a qualquer objeto, em tempo de execução, sem necessitar ter o conhecimento prévio das interfaces dos mesmos. Os objetos podem ser encontrados pelos serviços de nomes ou de *trading*.

A importância desta interface é que, caso um objeto necessite ser modificado constantemente, o sistema não precisará ser parado e recompilado toda vez que isto acontecer, como na invocação estática.

Para que um cliente emita uma requisição a um objeto, ele precisa dizer quem é o objeto-destino (através de sua referência), a operação que deseja que seja realizada e os parâmetros da operação. Tudo isto é especificado pela DII, que precisa realizar várias chamadas de rotinas de requisição, para que possa obter todos estes dados, que são recuperados nos IR específicos.

Estes acessos são transparentes, ou seja, não importa se o IR é local ou remoto. O problema é que pode ser acumulado muito *overhead*, caso vários acessos remotos necessitem ser feitos.

O cliente de posse da referência de objeto e dos parâmetros da operação, possibilita a DII, através das rotinas descritas abaixo, montar a requisição ao objeto-destino.

Rotina	Descrição
create_request ( )	cria uma requisição ORB, que será invocada por uma chamada à rotina **invoke ( )**, ou utilizando **send ( ) e get_response ( )**.
Add_arg ( )	adiciona argumentos incrementalmente a uma requisição
invoke ( )	chama o ORB, que faz a invocação do método apropriado, retornando o controle para o cliente após concluída a operação
delete ( )	apaga uma requisição que estava sendo montada
send ( )	inicia uma operação de modo assíncrono, retornando o controle para o cliente sem aguardar o término da operação
send_multiple_request( )	inicia várias requisições em paralelo, retornando o controle ao cliente sem aguardar que as operações sejam concluídas
get_response ( )	informa se uma requisição específica foi completada; dependendo dos *flags* passados em sua chamada, pode retornar imediatamente ou aguardar, caso a requisição ainda não tenha sido completada
get_next_response ( )	Informa se alguma requisição foi completada; dependendo dos *flags*, pode retornar imediatamente ou aguardar o fim de uma requisição, caso não haja nenhuma pendente.

Tabela 3.2 - Rotinas de Requisição da DII [13]

A requisição para que seja invocada, deverá primeiramente receber valores de argumentos (que poderão ser determinados a partir do IR), e então invocar operações diretamente sobre o pseudo-objeto *Request*. Em suma, isto significa que um pseudo-objeto será criado e valores de argumentos atribuídos à ele. A partir de então, existem três formas de se efetuar uma invocação:

1. Invocação Síncrona: ao emitir a requisição, o cliente bloqueia a espera até que a resposta chegue. Este é o tipo mais utilizado pelo CORBA, devido também ser suportado pelos *stubs* estáticos.

2. Invocação Síncrona Deferida: podem ser emitidas várias requisições e as respostas recebidas na medida que forem chegando.

3. Invocação *Oneway*: a requisição é emitida e não espera por uma resposta de volta.

As aplicações CORBA que necessitarem utilizar a *Invocação Síncrona Deferida*, deverão utilizar as facilidades da DII, uma vez que somente esta fornece tal serviço, por enquanto.

3.8.2. INTERFACE DE *SKELETON* DINÂMICA – DSI (*DINAMIC SKELETON INTERFACE*)

A DII permite que clientes emitam requisições sem a necessidade de acessar *stubs* estáticos. O mesmo acontece com a DSI, sendo que em prol dos servidores. Ela possibilita que os mesmos sejam escritos, sem que haja para isto, um *skeleton* para cada objeto a ser invocado.

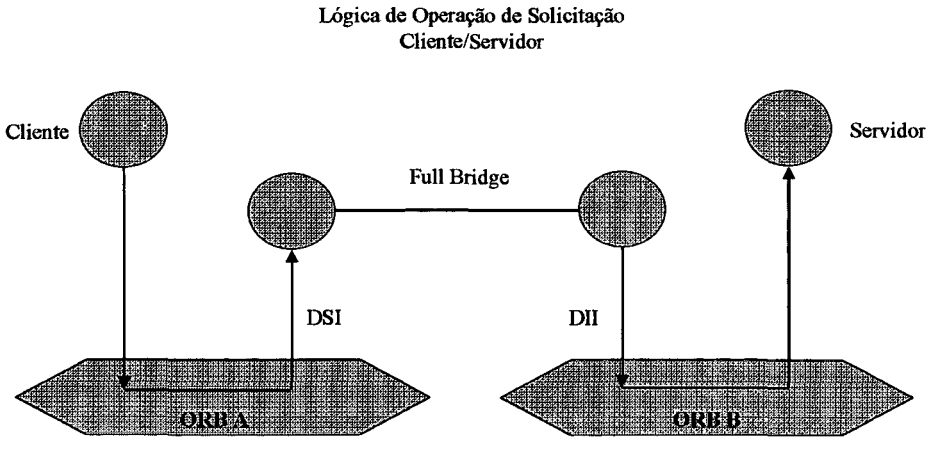

Figura 3.8 - Ilustração do Funcionamento de uma DSI [14]

Como mostra a figura, quando um cliente necessita acessar um serviço de um ORB remoto, o ORB do próximo cliente roteia a requisição à DSI, a qual serve de Interface *Skeleton* para o objeto no ORB remoto.

3.9. INTERFACE ORB

A interface ORB, utilizada tanto pelo cliente como pelo servidor, consiste de um pequeno conjunto de APIs (*Aplication Program Interface*) para serviços locais que podem ser de interesse de uma aplicação. Por exemplo, para a conversão de referências de objetos na forma de uma cadeia de caracteres e vice-versa. Isto pode ser útil se for necessário o intercâmbio de referências ou o armazenamento delas [1].

3.10. ADAPTADORES DE OBJETOS - OA (*OBJECT ADAPTER*)

O CORBA tem como objetivo integrar os mais variados tipos de objetos com várias linguagens de programação. Foi com este objetivo que foi criado o OA. Através dele, o ORB se comunica com a implementação do objeto, permanecendo o mesmo, o mais simples possível e ainda interagindo com vários tipos de objetos.

Um ORB pode possuir vários OAs, cabendo à implementação decidir qual irá utilizar.

Os OAs também podem ser específicos, isto é, um OA pode ser criado especificamente para suportar objetos com implementação em C e outro em SmalTalk.

O OA está localizado entre o ORB e a implementação e deverá auxiliar o ORB na entrega das requisições aos objetos, assim como ativá-los quando necessário.

Ele ainda possui outras atividades como:

1. Registro de Objetos: as entidades de linguagem de programação podem ser registradas como implementações de objetos CORBA, devido as operações supridas pelo OA;

2. Geração de Referência ao Objeto: referências a objetos CORBA são geradas pelo OA;

3. Ativação de Processos do Servidor: caso necessário, os OAs podem iniciar processos onde objetos podem ser ativados;

4. Ativação de Objetos: no caso de haver uma requisição para um determinado objeto que esteja desativado, o OA poderá ativá-lo;

5. Demultiplexação de Requisições: o OA deverá trabalhar junto com o ORB, de forma a garantir a entrega de requisições através de conexões múltiplas sem bloquear uma destas conexões de forma indefinida;

6. Chamadas a Objetos: os OAs são responsáveis por enviar requisições a objetos registrados.

Para tentar evitar a proliferação demasiada de OAs, desnecessária, o OMG criou o BOA (*Basic Object Adapter* – Adaptador de Objeto Básico), que é um adaptador específico do CORBA.

Contudo, devido à preocupação de fazer com que o BOA suportasse várias linguagens, outros aspectos deixaram muito a desejar, como registro de objetos de linguagens de programação, tais como objetos CORBA. Isto resultou em algumas incompatibilidades entre o BOA e os ORBs de vários vendedores.

A figura abaixo descreve o funcionamento do BOA através da demonstração dos passos da requisição de um cliente que chega no lado do servidor.

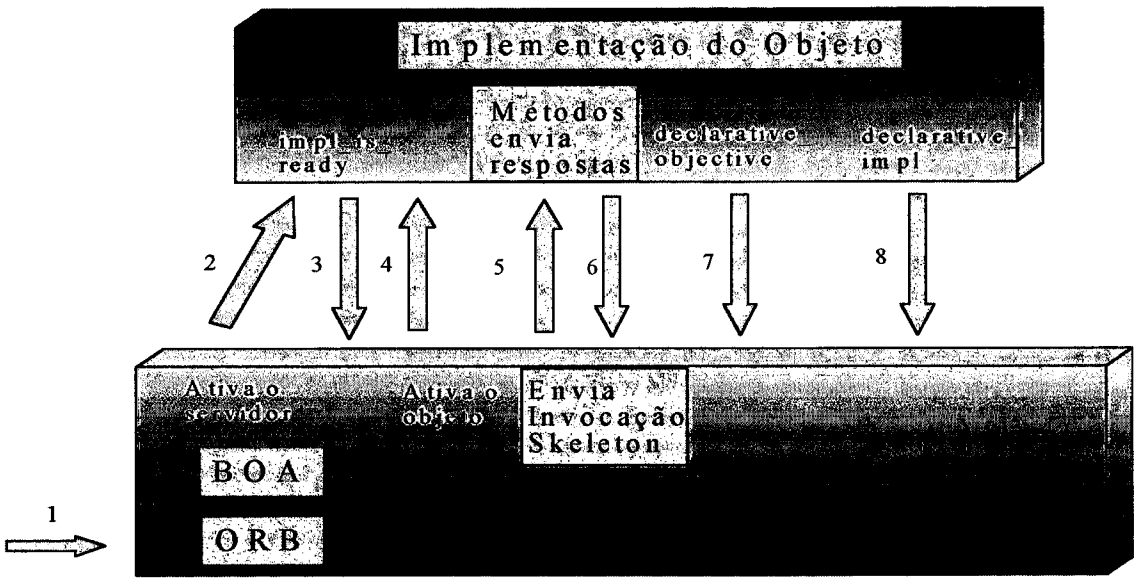

Figura 3.9 - Esquema do funcionamento do BOA [13]

1. O ORB, ao receber uma requisição para um determinado objeto, verifica se o servidor está ativo ou não;
2. Caso o servidor não esteja ativado, o ORB o ativará e passará toda a informação necessária para que este possa se comunicar com o BOA;
3. Quando o servidor estiver pronto, ele envia uma mensagem ***imp_is_ready*** ao BOA;
4. O BOA passa a referência ao objeto para a rotina de ativação do objeto-alvo, para que o servidor o ative;
5. O BOA passa a invocação ao objeto através do *skeleton* estático e logo que recebe a resposta de volta, a envia ao cliente;

6. Caso o BOA receba outras chamadas sobre este objeto, o passo 5 deverá ser repetido. E caso ele receba chamadas sobre outros objetos, ele deverá repetir os passos 4 e 5 para cada objeto;
7. Se o servidor necessitar desativar um objeto, ele deverá enviar uma chamada ***desactivate_obj*** ao BOA, informando que o objeto estará desativado, caso ele receba uma nova requisição;
8. Para que o servidor possa ser desativado, ele deve enviar uma chamada ***desactivate_impl*** ao BOA.

Para solucionar a problemática do BOA, foi criado o POA (*Portable Object Adapter* – Adaptador de Objeto Portátil), tornando o lado servidor do Corba tão portátil quanto o lado cliente da versão 2.0 do mesmo.

Os desenvolvedores do Corba decidiram construir um novo adaptador e objeto, pois tentar aprimorar o BOA poderia gerar problemas de incompatibilidades com os Orbs já existentes. Com isto surgiu o POA, que é uma versão portátil do BOA.

Aplicações já existentes não precisarão reescrever seus objetos do servidor, apenas os códigos que possuem interfaces diretamente ligadas ao ORB, necessitarão fazê-lo. Contudo, serão modificações bem simples que não acarretarão maiores problemas.

O POA permite que programadores de sistemas penetrem no código interno do servidor Corba, fazendo com que se possa controlar quase todos os aspectos do ambiente servidor deste.

Mesmo com a criação deste novo adaptador de objeto, o BOA não desaparecerá, garantem os vendedores (pelo menos por alguns anos ainda). O Orbs permitirão a existência de ambos, uma vez que foi desenvolvido para suportar vários tipos de adaptadores de objetos. Mas, aos poucos, a tendência é o desaparecimento do BOA.

3.11. INTEROPERABILIDADE

O CORBA para ser uma especificação completa, deveria possibilitar a interoperabilidade entre ORBs de diferentes fabricantes. Contudo, devido atuarem em ambientes heterogêneos, houve o problema com os diversos protocolos utilizados.

Tal problema, é solucionado pela interconexão de ORBs através de *bridges* (pontes).

No CORBA, as fronteiras de interoperabilidade são determinadas por Domínios. Domínio foi a forma que o OMG criou para agrupar os objetos que possuem características semelhantes, sejam elas relativas à implementação ou puramente administrativa.

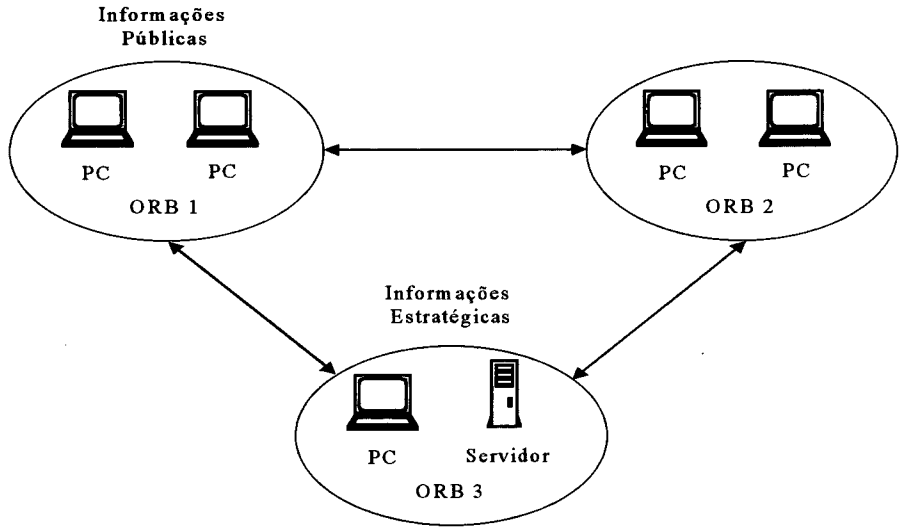

Figura 3.10 – Exemplo de Interoperabilidade entre ORBs [1]

Existem dois tipos de interoperabilidade entre ORBs no CORBA, a direta e a indireta.

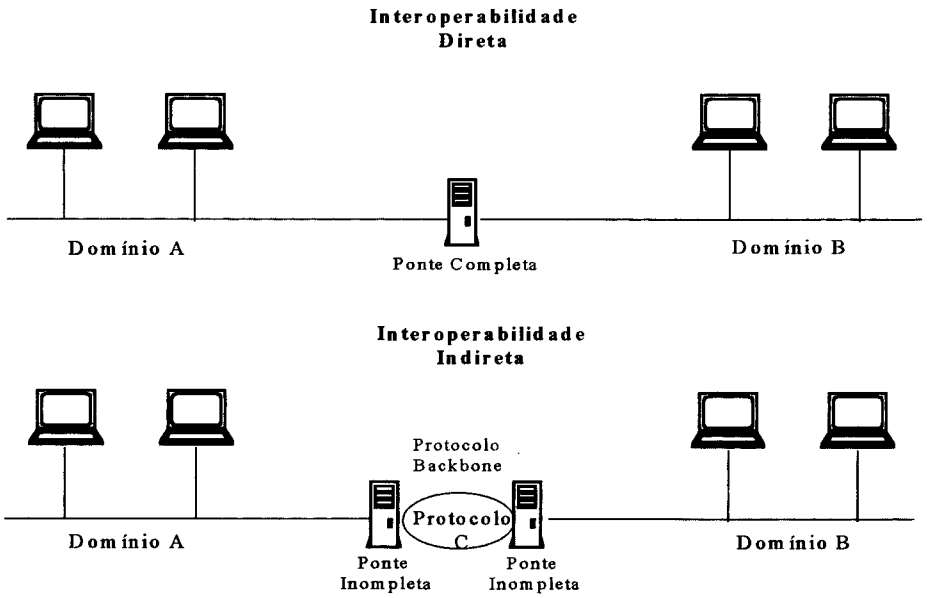

Figura 3.11 - Os dois tipos de interoperabilidade do CORBA [13]

3.11.1. INTEROPERABILIDADE DIRETA

Na direta, os domínios estão ligados diretamente através de *bridges* (pontes). Já na indireta, as *bridges* estão ligadas a um protocolo comum e não a outro ORB.

As *bridges* diretas ligam dois domínios, traduzindo de forma direta o protocolo de um para o protocolo do outro. Sua vantagem é que são bem rápidas e eficientes. Porém, são muito inflexíveis, pois deverá haver uma *bridge* para cada par de domínios. Logo, no caso de haver oito domínios, serão necessários 28 (vinte e oito) *bridges*.

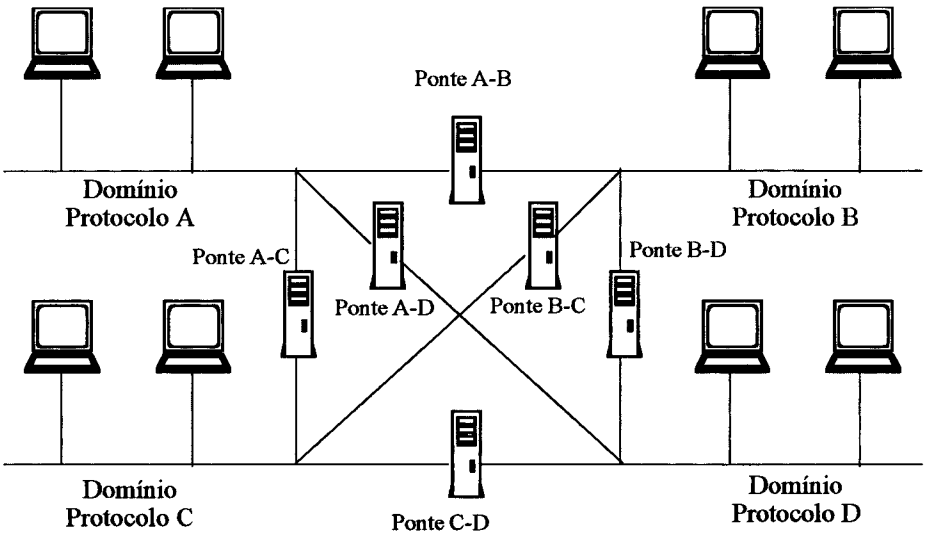

Figura 3.12 - Exemplo do uso de bridge direta com quatro domínios [13]

3.11.2. INTEROPERABILIDADE INDIRETA

As *bridges* indiretas são mais flexíveis, pois elas traduzem o protocolo de um domínio, primeiramente para um protocolo comum e a *bridge* do destinatário é quem traduz para o protocolo de tal domínio. Este protocolo comum tanto pode ser o IIOP (*Internet Inter-Orb Protocol* – Protocolo Inter-ORB da Internet), um padrão especificado pelo OMG, como um acordo privado entre as duas partes.

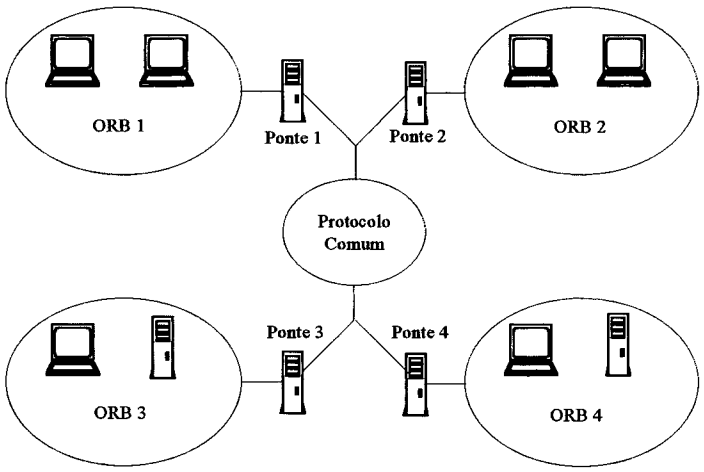

Figura 3.13 – Exemplo de Bridges Indiretas [13]

3.11.3. ORB INTERGALÁTICO

Na versão 1.1 do CORBA, a preocupação principal era a criação de aplicações de objetos portáveis. Criou-se então um certo nível de portabilidade dos componentes, porém não de interoperabilidade. Isto porque a implementação básica do ORB, era deixada a cargo de cada vendedor.

Um dos objetivos do CORBA 2.0, era o de implantar a interoperabilidade entre os vários ORBs de diferentes vendedores. Para isto foi especificado o IIOP, um protocolo obrigatório.

O IIOP funciona como um protocolo de *backbone* comum, que consiste essencialmente do TCP/IP e de trocas de mensagens já definidas do CORBA. Para que um ORB esteja dentro do padrão CORBA, ele precisa implementar o IIOP ou fornecer uma *half-bridge* (ponte incompleta) - é denominada assim devido o IIOP ser o padrão do CORBA - para este. Desta forma, ORBs de diferentes fabricantes poderão conectar-se, traduzindo as solicitações de e para o *backbone* IIOP [3].

No caso de redes específicas necessitarem interoperar, elas poderão utilizar o ESIOP *(Environment-Specific Inter-ORB Protocol* – Protocolo Inter-ORB para Ambiente Específico), também suportado pelo CORBA. O DCE é a primeira opção de ESIOP do CORBA 2.0, pois fornece um ambiente robusto para ORBs de missão crítica.

Os ORBs podem ser organizados em domínios, que podem ser segundo as necessidades administrativas, implementações do ORB de um vendedor, protocolos de segurança e outros.

ORBs federados permitem a criação de topologias bem flexíveis, através da utilização de pontes inter-ORBs e dos IIOPs.

3.11.4. ARQUITETURA INTER-ORB

Desenvolvedores de aplicações não necessitam se preocupar em saber como é que ocorre a interoperabilidade dos ORBs. Raros são os casos em que isso ocorre. Contudo, como o objetivo deste trabalho é possibilitar uma avaliação de onde e como a plataforma CORBA pode ser melhor aplicada, então será apresentado neste item, uma visão geral dos elementos que compõe esta arquitetura.

Para que a solução através de *bridges* fosse possível, era necessário haver um padrão para a sintaxe de transmissão. Tal padrão foi definido pelo OMG e está descrito no Protocolo Geral Inter-Orb (GIOP).

A figura 3.14 mostra os elementos do CORBA2.0 na arquitetura inter-ORB.

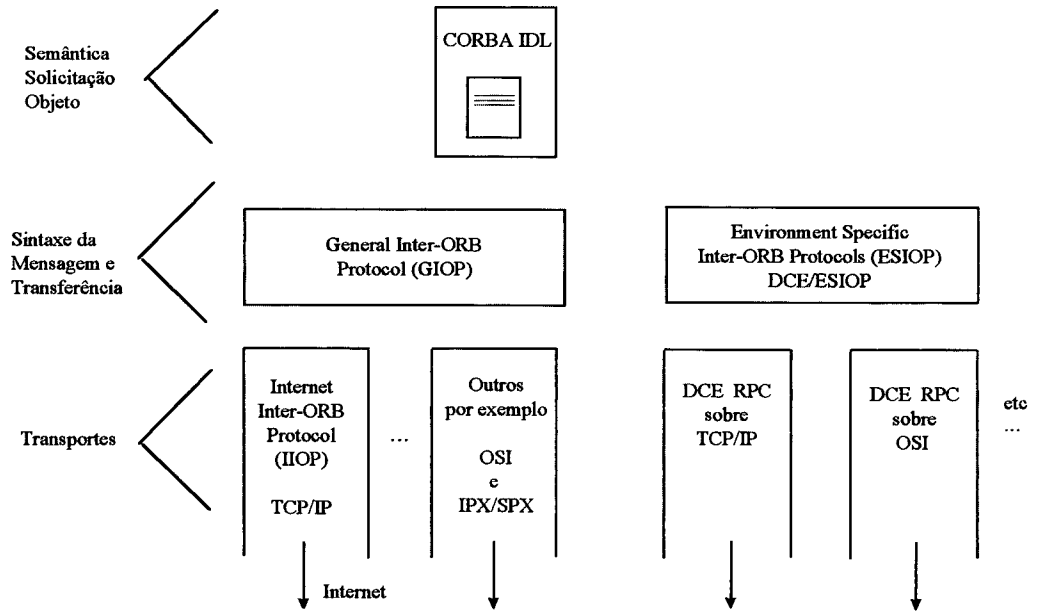

Figura 3.14 - Arquitetura Inter-ORB do CORBA 2.0 [13]

Primeiramente, será dado o conceito de IOR (*Interoperable Object References* – Referências de Objetos Interoperáveis), para depois então, abordar os conceitos dos elementos da figura.

O IOR é um formato definido pelo GIOP. Um ORB deve criar um IOR toda vez que uma referência a objeto é passada através dos ORBs. Os IORs associam uma coleção de

rótulos às referências a objetos. Os rótulos descrevem o mesmo objeto, mas cada um descreve um meio para contatar o objeto utilizando um mecanismo de ORB particular [1] [13].

- General Inter-ORB Protocol (GIOP): foi projetado para trabalhar diretamente sobre protocolos de transporte orientados à conexão. Na maioria dos casos, os clientes enviam uma solicitação para objetos imediatamente após ser estabelecida a conexão. Tal protocolo especifica um grupo de formatos de mensagens e representações de dados comuns para a comunicação entre ORBs. O GIOP abrange todos os formatos de mensagens utilizados nas semânticas de solicitação e resposta dos ORBs, portanto, não há a necessidade de negociação de formatos. O CDR (*Common Data Representation* – Representação de Dados Comuns) é responsável por mapear os tipos de dados definidos em OMG IDL para uma representação de mensagem da rede, além de questões inter-plataforma, como a ordenação de bytes e alinhamentos de memória.

- Internet Inter-ORB Protocol (IIOP): o TCP/IP foi o protocolo escolhido pelo OMG para implementar o GIOP, devido ser o protocolo de transporte mais difundido. O protocolo é o IIOP. Este especifica como as mensagens do GIOP são trocadas através de uma rede TCP/IP. Sua estrutura deve ser simples e capaz de fornecer interoperabilidade "fora da caixa", isto é, com outros ORBs compatíveis baseados no TCP/IP. Para que o CORBA 2.0 seja compatível, um ORB deve suportar o GIOP sobre TCP/IP ou conctar-se a este via uma *half-bridge*.

- Environment-Specific Inter-ORB Protocols (ESIOPs): são utilizados para interoperabilidade "fora da caixa" sobre redes específicas. O CORBA 2.0 especifica o DCE (*Distirbuted Computing Environment* – Ambiente para Computação Distribuída) como o primeiro de várias opções de ESIOPs. Tal como o GIOP, o DCE/ESIOP, suporta o IOR, usando um perfil do próprio DCE. Ele também, fornece um ambiente robusto para ORBs de missão crítica. Isto inclui características avançadas, tais como segurança em Kerberos, células e diretórios globais, tempo distribuído e RPC autenticadas. O DCE também permite que grandes quantidades de dados sejam transmitidos eficientemente e suporta múltiplos protocolos de transporte subordinados, incluindo TCP/IP. O DCE também pode ser usado tanto com protocolos com conexão quanto sem conexão.

3.11.5. ORBs Federados

As pontes inter-ORB e os IIOPs, podem ser criados para criar topologias muito flexíveis através da federação dos ORBs. A figura 15 mostra um *backbone* IIOP com vários ORBs proprietários, que serão transformados nesse (IIOP) através de *half-bridges*. Pode-se segmentar os ORBs em domínios baseados em necessidades administrativas, implementações de ORBs por vendedor, protocolos de rede, cargas de tráfego, tipos de serviço e conceitos de segurança. Pode haver conflito de políticas entre as partes, então *firewalls* podem ser criados em torno do *backbone* do ORB através das *half-bridges*. O CORBA 2.0 promove diversidade e fornece total flexibilidade, contanto que o IIOP seja utilizado para o *backbone* global [1] [13].

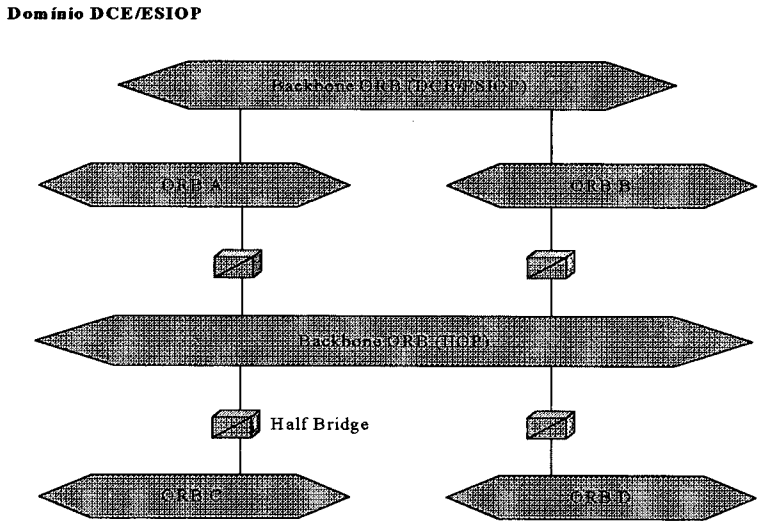

Figura 3.15 - Uma Federação Intergalática de Multivendedores ORBs [13]

3.12. Considerações Finais

O CORBA foi desenvolvido com o objetivo de utilizar, da melhor maneira possível, a O.O. Portanto, suas vantagens vão desde a facilidade para criação dos seus objetos, até os benefícios que a interconexão dos mesmos proporciona.

Abaixo estão descritas algumas das vantagens do CORBA:

1. Reutilização de todas as ferramentas adquiridas: esta reutilização abrange tanto hardware quanto software, não importando se são compatíveis, ou não.

2. Facilidade de conexão de componentes novos ou já existentes: para que um novo componente seja incluído, ou para reformular um já existente, basta acrescentar uma interface IDL a cada componente.

3. Maximiza a produção da equipe de desenvolvimento: as suas características de acesso transparente, portabilidade entre plataformas, orientação a objeto e serviços pré-definidos, faz com que os desenvolvedores possam se preocupar somente com a aplicação, esquecendo detalhes de funcionamento.

4. Reutilização de códigos e objetos: devido utilizar conceitos da O.O., ele permite a reutilização de códigos já existente.

5. Melhor aproveitamento dos objetos por parte de usuários e clientes: os objetos CORBA podem ser utilizados por vários usuários em diversos setores da empresa, ou mesmo entre duas companhias, não sendo necessário comprá-los ou reinstalá-los.

Em resumo, o CORBA é um produto que tenta se adequar às necessidades do cliente ao invés de tentar fazer com que este se adeqüe a ele.

IV CAPÍTULO
DCOM
(DISTRIBUTED COMPONENT OBJECT MODEL)

4.1. Arquitetura DCOM

O DCOM foi criado pela Microsoft como uma extensão do COM (*Component Object Model*), com o objetivo de suportar a comunicação entre objetos em diferentes computadores.

Uma vez que o DCOM é uma evolução do COM, ele pode tirar proveito do mesmo e fazer uso de investimentos já existentes deste, tais como aplicações, componentes, ferramentas e conhecimento para ingressar no mundo da computação distribuída baseado nas padronizações existentes. Desta forma, o DCOM tratará os detalhes de baixo nível dos protocolos de rede, fazendo com que os desenvolvedores possam se preocupar com tarefas mais importantes, tal como fornecer melhores soluções para seus clientes.

O COM define como deve ser a interação entre os objetos e seus clientes, sem a intermediação de qualquer componente do sistema.

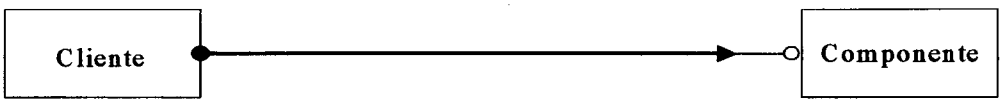

Figura 4.1. Componentes COM no mesmo processo [24]

No caso de componentes que estejam sendo executados em diferentes processos, existe a necessidade de ser estabelecida uma comunicação entre processos, a qual é feita pelo sistema operacional. O COM fornece esta comunicação através da biblioteca de execução COM/DCOM que estabelece o *link* entre o cliente e o componente.

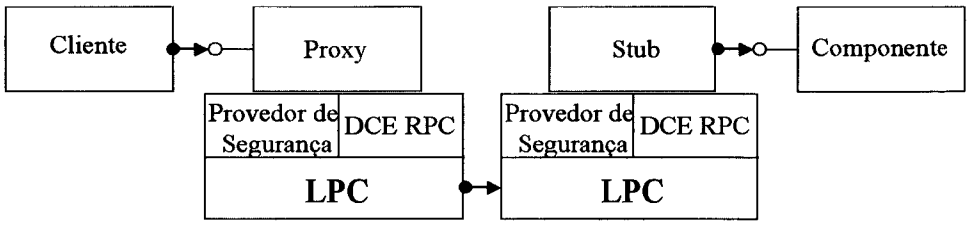

Figura 4.2. Componentes COM em diferentes processos [24]

Quando o processo cliente e o componente estão localizados remotamente, o DCOM utiliza um protocolo de rede para estabelecer a comunicação entre eles de forma transparente. O *proxy* e o *stub* fornecem serviços orientados a objetos ao cliente e ao componente, respectivamente, utilizando o RPC (*remote procedure call* – chamada remota a procedimento) e o Provedor de Segurança para gerar pacotes de redes padrões que sejam compatíveis com o protocolo padrão do DCOM.

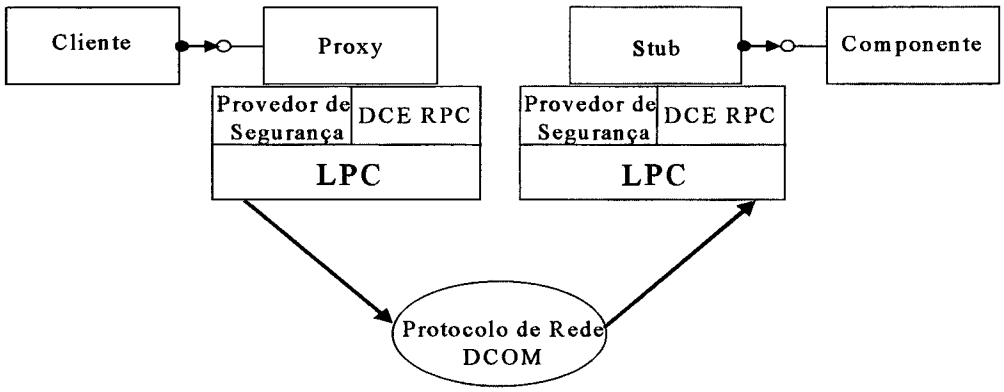

Figura 4.3. DCOM – Componentes COM em diferentes máquinas [24]

No momento da implementação de uma aplicação distribuída em uma rede, surgem vários problemas de conflitos relacionados ao projeto, tais como:

- Componentes que interagem mais, deveriam estar mais próximo um do outro;
- Alguns componentes só podem ser executados em máquinas específicas ou em localizações específicas;
- Componentes menores aumentam a flexibilidade de desenvolvimento, mas eles também aumentam o tráfego na rede;
- Componentes maiores reduzem o tráfego, porém também reduzem a flexibilidade.

No DCOM estes problemas são facilmente contornados, pois detalhes de desenvolvimento não são especificados no código fonte. A localização dos componentes é um fator totalmente transparente. Não importa onde eles estejam localizados, a forma como são feitas as chamadas é a mesma tanto para os localizados no mesmo processo, quanto para os que estão do outro lado do mundo. Uma simples reconfiguração altera o modo como os

componentes são conectados uns aos outros, sem a necessidade de haver mudanças no código fonte e nem de uma recompilação do programa.

4.2. Definição dos Objetos e Interfaces DCOM

A IDL é utilizada pelo DCOM para declarar as interfaces dos objetos, as quais não são declaradas junto com a sua implementação.

O conceito de herança de interfaces não é suportado pelo DCOM, porém, seus componentes podem suportar múltiplas interfaces e ativar a reusabilidade através do encapsulamento dos componentes internos das interfaces e representação dos mesmos ao cliente.

Um objeto pode suportar várias interfaces de acordo com a sua classe. Esta classe é identificada por um valor único de 128 bits, representado pelo CLSID (*Class* ID). Os objetos são instâncias em tempo de execução da classe. Por isso a palavra "objeto" pode ser usada para se referir tanto a classe do objeto, quanto a uma instância individual da classe [3].

As interfaces são um grupo de funcionalidades de métodos relacionados, as quais são implementadas por um determinado objeto da sua classe correspondente.

Os objetos DCOM não possuem um identificador único e para que eles sejam acessados, os clientes necessitam utilizar os ponteiros de interface. Um cliente não poderá se reconectar ao mesmo objeto mais tarde com o mesmo estado, ele poderá apenas reconectar-se com o ponteiro de uma interface da mesma classe. Isto ocorre, pois os objetos DCOM não mantêm um estado entre conexões, o que pode ser um problema em ambientes com falhas de conexões, como a Internet [3] [18].

Para que um cliente acesse uma interface, ele deve utilizar ponteiros para um vetor de funções de ponteiros, chamados de tabela virtual (*virtual table – vtable*). Estas funções correspondem aos métodos de implementações dos objetos do servidor. Cada objeto pode possuir uma ou mais tabelas virtuais que definem o contrato entre a implementação do objeto e seus clientes. A figura 4.4 descreve como isto é feito [3] [18].

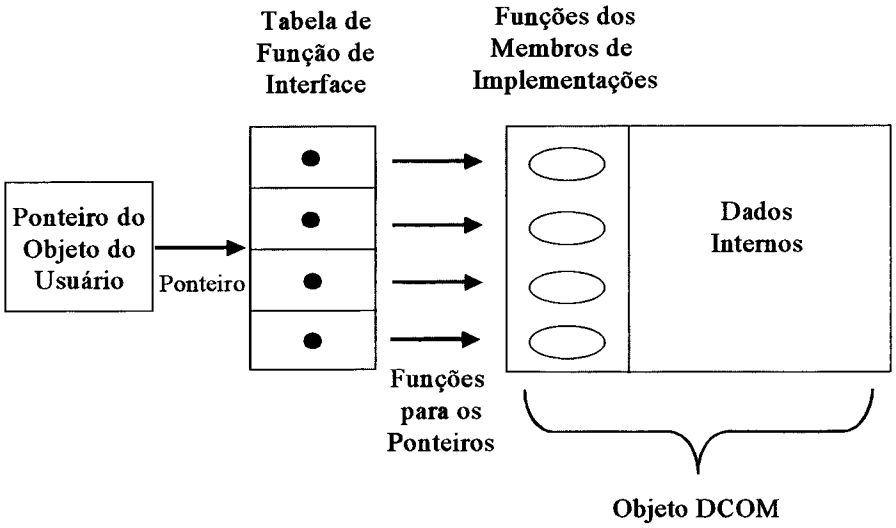

Figura 4.4. Ponteiro para uma Representação de uma Tabela Virtual [3]

4.3. SERVIDOR DCOM

Este servidor é um código, que pode ser uma DLL, um EXE ou uma classe Java, que pode suportar várias classes de objetos, cada uma com seu CLSID e que irá criar um objeto de uma determinada classe toda vez que for solicitado. O que ocorre é o seguinte, toda vez que um cliente solicitar por um objeto de uma classe específica, o DCOM deverá carregar o servidor e requisitar a este que crie um objeto desta classe. Uma classe denominada *factory* deve ser fornecida pelo servidor para a criação do mesmo. Um ponteiro para a interface primária do objeto será retornado ao cliente após a criação deste.

A tabela 1 mostra o que o servidor deve fazer para tornar um objeto acessível aos clientes.

AÇÃO	Descrição
Implementar uma interface para a classe *factory*	Para cada CLSID suportada, o servidor deverá implementar uma classe *factory* com uma interface ***IclassFactory***. Caso a classe aceite permissões, então uma interface ***IClassFactory2*** deverá ser implementada e somente os objetos que possuam um arquivo de permissão ou uma senha, serão criados.
REGISTRAR A CLASSE SUPORTADA	Para cada classe que o servidor suporta, ele deve registrar uma CLSID. Geralmente as classes são registradas no momento da instalação.
INICIALIZAR A BIBLIOTECA DO DCOM	A biblioteca fornece serviços em tempo de execução e APIs para que ela seja inicializada. O servidor precisa enviar um chamado *CoInitialize* para a API do DCOM.
Verificar se a biblioteca é de uma versão compatível	Esta verificação é feita através da chamada da API *CoBuildVersion*.
Fornecer um mecanismo de finalização do próprio servidor	O servidor deverá finalizar a si próprio quando não houver nenhum cliente ativo para os seus objetos.
Finalizar a biblioteca do DCOM	A API *CoUninitialize* é chamada para a finalização da biblioteca.

Tabela 4.1. Estrutura do Servidor para Disponibilizar um Objeto [3]

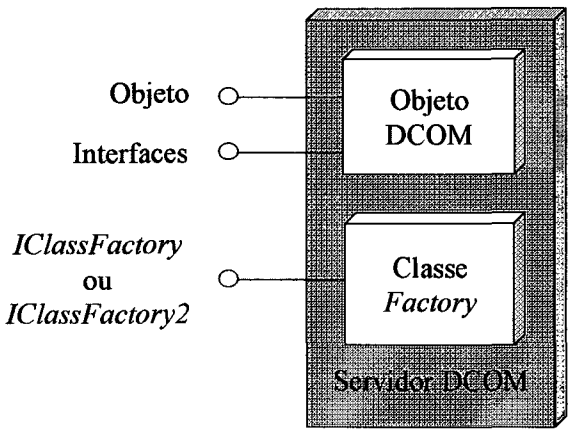

Figura 4.5. Estrutura de um Servidor DCOM [3]

O servidor pode ser implementado de várias formas [3] [18]:

- **Servidores *In-process***: quando o servidor estiver sendo executado no mesmo processo que o cliente. No ambiente Windows e Windows NT, estes servidores são implementados como DLLs (*Dynamic Link Libraries*) e são carregados diretamente no processo cliente.
- **Servidores Locais**: são executados em processos diferentes, porém na mesma máquina. O LRPC (*Lightweight RPC*) do DCOM é utilizado pelos clientes para a comunicação com estes servidores.
- **Servidores Remotos**: são executados em processos e máquinas diferentes, podendo ser também em outro sistema operacional. Para a comunicação com estes servidores, os clientes utilizam o mecanismo de RPC.

Tanto para os clientes quanto para os servidores, toda esta comunicação é feita de forma transparente. Para os clientes, todos os acessos são feitos através de ponteiros de interfaces, que de uma forma ou de outra, estarão em algum momento no mesmo processo que o servidor, ou *in-process*, que significa a mesma coisa. Quando o objeto já está *in-process*, a chamada automaticamente também já está neste processo. Porém, se o objeto estiver *out-of-process*, isto é, em um processo ou máquina diferente do servidor, sua solicitação deverá ser passada primeiramente para o *proxy* (corresponde ao *stub* do cliente no CORBA), que irá gerar o mecanismo de RPC apropriado e transmití-la ao processo ou máquina destino. O SCM ou "Scum" (*Service Control Manager* - Gerente de Controle de Serviço), é o elemento responsável por localizar o servidor, além de ser quem se envolve com as invocações de RPC entre clientes e servidores.

No caso dos servidores, é o processo requisitor que em algum momento estará *in-process*. Se o objeto já estiver no mesmo processo, então é porque este é o próprio cliente. Caso contrário, este objeto é o *stub* (corresponde ao *skeleton* no CORBA), que recebeu a mensagem remota do *proxy* e transformou-a em uma chamada de interface para o objeto do servidor.

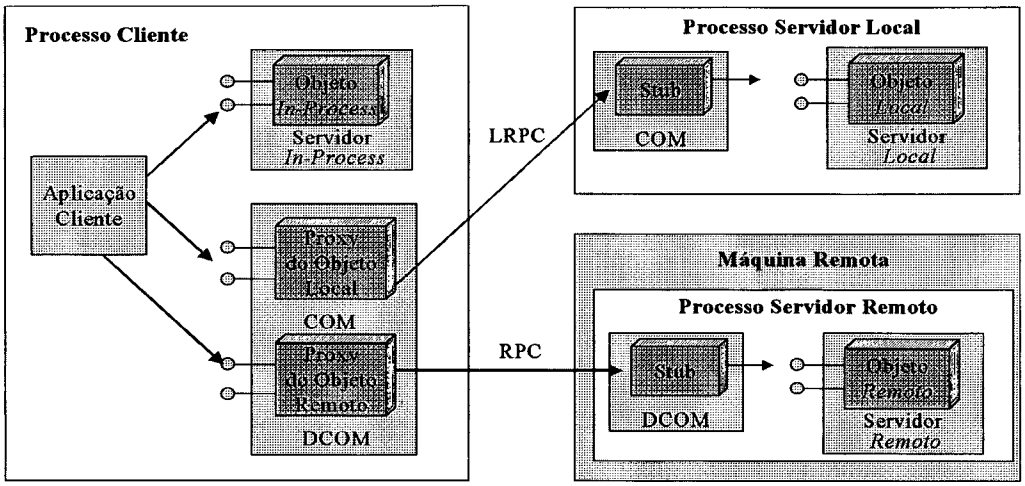

Figura 4.6. Limites dos Cliente/Servidores do DCOM [18]

4.4. UBIQUITOUS IUNKNOWN INTERFACE

No DCOM, todas as interfaces derivam da interface *IUnknown*. É através desta que são feitas as negociações de interface em tempo real, gerenciamento de ciclo de vida e agregação. As funções implementadas por ela, e herdadas pelas demais, são: *QueryInterface*, *AddRef* e *Release* [3].

- ***QueryInterface***

Esta função é responsável pela negociação de interfaces e agregação. Na negociação, um cliente, ao obter acesso a um objeto, recebe um único ponteiro de interface. A partir daí, através desta função, ele pode obter as demais interfaces que são suportadas pelo objeto em tempo de execução.

Um objeto também pode recusar um pedido do cliente, basta que ele não retorne um ponteiro. Neste caso, e também se a interface não for suportada pelo servidor, a função *QueryInterface* retorna um código de falha.

Para realizar a agregação, esta função tem a capacidade de retornar ponteiros para as outras interfaces que um objeto DCOM suportar. Toda vez que uma chamada para uma interface específica for realizada, a função *QueryInterface* deverá retornar o valor real do ponteiro, não importando através de qual interface derivada da *IUnknown,* a chamada foi feita.

Novas interfaces podem ser adicionadas pelos componentes e disponibilizadas aos clientes através desta mesma função. Desta forma, clientes podem interrogar um objeto para saber se ele suporta determinada interface e obter como resposta tanto um "não", de um objeto antigo, quanto um "sim" de um objeto mais novo. Isto permite a compatibilidade entre clientes e objetos criados antes e depois da nova interface. Esta negociação dinâmica é uma grande vantagem de controle da versão DCOM.

- ***AddRef* e *Release***

Estas duas funções foram criadas com a finalidade de controlar o ciclo de vida dos objetos, de forma que objetos que não estiverem mais sendo utilizados sejam descarregados. Isto é feito da seguinte forma: quando um cliente solicitar um ponteiro de uma interface, ou passá-lo adiante, a função *AddRef* é chamada e o contador de referência da interface é incrementado. Da mesma forma, quando o cliente deixar de utilizar a interface, a função *Release* é chamada e o contador decrementado. No momento em que o contador atinge 0 (zero), o objeto é descarregado.

Esta forma de controle do ciclo de vida dos objetos permite componentes obter e disponibilizar o acesso a um único objeto sem a necessidade de coordená-lo com o ciclo de vida de outros objetos. Contudo, há uma carga extra que é adicionada a implementação de todos os objetos, além de que há a necessidade da cooperação dos clientes, o que não é uma grande garantia.

4.5. ICLASSFACTORY2 : CRIAÇÃO DE OBJETO E PERMISSÕES

Esta interface é uma extensão da interface *IClassFactory*, isto é, possui todos os seus métodos e mais alguns, e ambas fazem parte da interface *class factory*, que deve ser implementada por todas as demais classes, para que o servidor possa chamá-la para criar instâncias destas classes.

Os métodos adicionais desta classe são o *GetLicInfo, RequestLicKey* e *CreateInstanceLic,* utilizados para a criação de licenças para os clientes e objetos, e controle das mesmas, que é o objetivo desta interface. A tabela 2 mostra todos os métodos contidos nesta interface e em seguida será feita uma descrição da funcionalidade de cada um.

IClassFactory2

Métodos da Interface *IUnknown*	Métodos da Interface *IclassFactory*	Novos Métodos da Interface *IClassFactory2*
QueryInterface	*CreateInstance*	*GetLicInfo*
AddRef	*LockServer*	*RequestLicKey*
Release		*CreateInstanceLic*

Tabela 4.2. Interface *IClassFactory*

O objetivo dos métodos da interface *IUnknown* já foram descritos no tópico anterior, uma vez que esta é a interface base de todas as demais interfaces do Dcom, inclusive *IClassFactory2.*

O servidor DCOM ao ser carregado, instancia uma *class factory* para cada uma das suas classes e as registra com o DCOM através da chamada da API *CoRegisterClassObject.*

Existem dois tipos de licença: o arquivo de licença, que é uma licença global que permite o acesso a todos os componentes de uma máquina; e a licença específica, utilizada para acessar um componente a partir de uma outra máquina.

Uma máquina é completamente licenciada se houver um arquivo de licença instalado na mesma. Se não, é necessário que o cliente ao instanciar um componente, forneça uma licença própria, ou o servidor não poderá instanciá-lo.

Os métodos *CreateInstance* e *CreateInstanceLic* são utilizados para criar uma nova instância de uma classe. Eles criam uma instância não inicializada do objeto associado com a *class factory* e retornam um ponteiro da interface solicitada. A diferença entre estes dois métodos é quanto ao tipo de licença. O *CreateInstance* dever ser utilizado quando houver um arquivo de licença instalado. Já o *CreateInstanceLic,* é utilizado pelo cliente quando não

houver este arquivo de licença e ele necessitar de um licença específica, que deverá ser enviada como um de seus argumentos.

O *LockServer* é utilizado para que um cliente mantenha um objeto do servidor ativo, mesmo que o seu contador de referência seja 0 (zero).

Utiliza-se o método *GetLicInfo* para a verificação do tipo de licença suportada pelo componente. E o *RequestLicKey*, para que um cliente solicite uma licença para o uso de um componente em uma máquina diferente.

A implementação do *IClassFactory2* deve validar a licença, seja ela do tipo que for, antes de criar uma instância de um componente.

Este tipo de validação de permissão não é muito eficaz, uma vez que o arquivo de licença é apenas um texto instalado no próprio componente e a licença específica, a primeira linha deste texto. Não há proteção contra invasores e nem como controlar o uso de licenças.

4.6. Estilo de Herança do DCOM: Agregação e Contenção

Embora o DCOM não implemente a herança múltipla, ao contrário do CORBA, seus componentes podem suportar várias interfaces através de dois métodos de encapsulação, que são agregação e contenção. Através das chamadas do método *QueryInterface*, um cliente é capaz de saber, em tempo real, todas as interfaces suportadas por um componente. Isto permite que desenvolvedores criem componentes externos que podem encapsular os serviços dos componentes internos e apresentá-los ao cliente.

- Agregação

  Quando um cliente deseja se conectar a um componente interno, o componente externo repassa o ponteiro da interface do objeto interno para ele, e com isto fica parecendo que a interface é a sua própria.

- Contenção

  Diferente da agregação, este método não estabelece o contato direto entre o cliente e o objeto interno. Ao invés disto, ele repassa as invocações recebidas pelos componentes externos, aos internos, quando estes forem os destinatários.

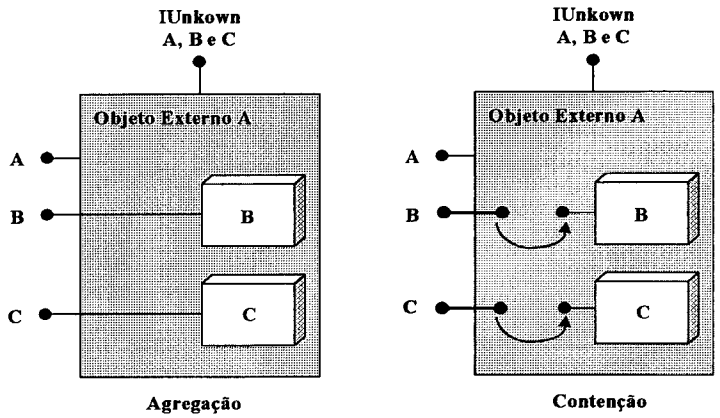

Figura 4.7. Agregação e Contenção no DCOM [3]

Através deste dois métodos de encapsulamento, a reusabilidade é aplicada no DCOM. Mesmo não implementando a herança de interfaces, ele suporta múltiplas interfaces, através de uma cadeia de ponteiros que ligam ou agregam diferentes interfaces.

4.7. Linguagem de Definição de Interface - IDL (*Interface Definition Language*) e Linguagem de Definição de Objeto - ODL (*Object Definition Language*) do DCOM

O DCOM, diferentemente ao CORBA, implementa duas linguagens de definição, a IDL e a ODL. A primeira é utilizada para gerar *stubs* e *proxies* remotos, enquanto que a segunda é para gerar os metadados das classes. Porém, atualmente, ambas utilizam o mesmo compilador, o MIDL, uma vez que a ODL é considerada como um subconjunto da IDL. E os arquivos ODL, podem inclusive, ser embutidos nos arquivos IDL.

A ODL é uma linguagem mais neutra que a IDL. Ela permite que a sintaxe usada para descrever os objetos seja mais geral. Através dela, podem ser descritas interfaces dinâmicas e estáticas, além de toda a estrutura de uma classe DCOM. A compilação de um arquivo ODL, o torna um arquivo tipo biblioteca (.TLB) para as interfaces desta classe.

Uma vez que um arquivo ODL pode estar dentro de um arquivo IDL, é necessário que o MIDL possua uma forma para realizar este reconhecimento. Isto é possível através da palavra *library*, que é o elemento que indica a declaração de uma sintaxe ODL.

Os arquivos em IDL podem possuir várias definições de interface, sendo que cada uma deverá conter um cabeçalho e um corpo.

4.8. INVOCAÇÃO DINÂMICA

A invocação dinâmica é a base para a automação, pois a partir dela, clientes podem invocar métodos que manipulam o conteúdo de objetos dinamicamente.

O CORBA implementa a invocação dinâmica através do seu Repositório de Interface. Já o DCOM o faz através da "*Type Library*". Ela permite aos clientes descobrirem os métodos e as propriedades, disponibilizadas por um servidor de automação, dinamicamente [3].

Os objetos de automação devem ser descritos pelos programadores e, esta descrição deve conter os nomes dos métodos, os tipos de seus parâmetros e as propriedades dos objetos. Estas interfaces devem ser declaradas utilizando a ODL.

Os servidores de automação são objetos que implementam uma interface denominada *IDispatch*. É através desta interface que os objetos disponibilizam as suas funções, as quais são chamadas de *dispatch interfaces* ,ou *dispinterfaces*. Tanto os métodos, quanto as interfaces e as propriedades dos servidores, são mantidas nas *type libraries*, que funcionam como um repositório de tempo real.

Existe uma interface, *ITypeLib*, que permite que os controladores de automação, isto é, os clientes, aprendam os nomes dos métodos e as propriedades que um objeto suporta. Eles também podem, através do método *IDispatch::Invoke*, criar ou obter objetos de automação, recuperar e especificar as propriedades que eles disponibilizam e executar seus métodos.

4.9. CONSIDERAÇÕES FINAIS

O Dcom foi idealizado visando a integração de ambientes heterogêneos, de forma a permitir a reutilização de ferramentas e objetos, diminuindo assim custos de novas aquisições e produções e também para um maior aproveitamento da produção da equipe de desenvolvimento.

Tal objetivo foi alcançado com sucesso sendo que através das plataformas Windows. Objetos podem migrar de um Windows 95 para um Windows NT sem maiores dificuldades. Já em relação a integração de plataformas heterogêneas não aconteceu o mesmo. O que é possível de ser feito neste caso, é com o auxílio de outros softwares fazer com que um sistema Unix por exemplo, se torne cliente do NT, contudo o inverso não é verdadeiro.

Constatou-se portanto, que o Dcom é uma tecnologia propícia para a interação de sistemas baseados na plataforma Windows que não necessitam interagir com objetos localizados em sistemas com outros tipos de plataformas. Apesar da desvantagem de não ter

conseguido atingir o seu objetivo inicial, para a integração de sistemas baseados em Windows o Dcom é uma boa alternativa, e dependendo do caso, melhor até que o Corba.

V CAPÍTULO
CORBA X DCOM

Os dois capítulos anteriores tratam das tecnologias Corba e Dcom, apresentando toda a sua estrutura. Eles abrangem seus principais conceitos, elementos e funcionamento. Toda esta abordagem serve de base para atingir o principal objetivo deste trabalho, que é o estudo comparativo entre estas plataformas. Esta comparação, neste trabalho, será abordada em duas partes: uma teórica, que é a que será vista neste capítulo e outra prática, abordada no capítulo seguinte.

O Dcom (*Distibuted Component Object Model)* foi criado pela Microsoft como sendo uma variação do COM (*Component Object Model*) para a integração de objetos distribuídos. Ele foi desenvolvido para ser primeiramente uma solução para a plataforma windows. Através da utilização do Software AG, o Dcom pode ser executado em várias plataformas UNIX. Contudo, ele necessita do MTS (*Microsoft Transaction Server)* para executar o servidor, o qual não é carregado pelo AG. Sendo assim, o UNIX torna-se um cliente do servidor NT, mas o oposto não ocorre. Tal tipo de problema não existe com o Corba, pois este é uma plataforma aberta que pode ser executada em qualquer sistema operacional, inclusive windows [13].

Outro fator que diferencia estes dois modelos para integração de objetos distribuídos, é a questão do acesso aos objetos. No Corba, cada objeto possui a sua interface e um identificador único. Já no Dcom, as interfaces não possuem estado e não podem ser instanciadas para criar um único objeto. Isto faz com que os clientes não possam se reconectar ao mesmo objeto, com o mesmo estado, outras vezes. Para ambientes que possuem muitas falhas de conexão, como a *Internet,* isto pode ser um problema.

5.1. DIFERENÇAS ENTRE DCOM E CORBA

5.1.1. ANÁLISE DAS DIFERENÇAS SOBRE FATORES ISOLADOS

Existem muitos fatores diferenciando estas duas plataformas. A tabela 5.1 apresenta uma visão geral de vários aspectos, que refletem várias diferenças entre estas duas plataformas. A comparação aqui realizada, não considera os requisitos necessários de um ambiente próprio, para a implementação destas plataformas. A avaliação é feita sobre aspectos isolados.

Aspectos	CORBA	DCOM
Integração com o Java	Utiliza interfaces Java para exportar seus serviços, fornecendo um ótimo ambiente de programação distribuída para Java.	O DCOM utiliza sua própria IDL, a qual ele integra com o modelo de Objeto Java.
Suporte para as Plataformas dos Sistemas Operacionais	Pode ser executado na maioria dos S.O. Cliente/Servidor.	Só pode ser executado, em primeiro plano, no NT, no Windows 95 e 98. Através da utilização do Software AG, ele pode ser executado em alguns sistemas UNIX.
Solução para Java	Existem implementações comerciais, todas em java do CORBA.	É necessário fornecer adaptadores java para o DCOM através do seu Java VM no Windows 95 e NT, uma vez que este é escrito em C e C++. O DCOM não pode ser executado em outros Sistemas Operacionais Java.
Invocações de métodos distribuídos	Possibilita que Objetos sejam reativados de acordo com o desejado, uma vez que suporta referências a Objetos únicas, obtendo assim melhor desempenho.	Permite tais tipos de invocações também. Contudo, devido seus Objetos não possuírem referências únicas, sua performance não é tão boa quanto a do CORBA.
Manter o estado através das invocações	Devido o CORBA sustentar a idéia de Objeto único e estado de Objeto, estes podem ser reconectados tempos depois, a partir do mesmo estado.	Mantém o estado de um Objeto dentro de uma sessão. Porém como não suporta o conceito de Objeto único, não pode manter o estado através das sessões.
Descoberta dinâmica	Permite tanto que interfaces de Objetos sejam descobertas dinamicamente, quanto os Repositórios de Interface inter –ORB.	Suporta a descoberta de interfaces de Objetos dinamicamente e repositórios locais para os tipos de biblioteca.
Segurança a nível de protocolo	O serviço de segurança do CORBA, foi implantado dentro da estrutura de mensagem do ORB e do IIOP, podendo trabalhar também com SSL.	O nível de segurança do NT foi implantado no DCOM e o do DCE também deverá ser no futuro.
Transações a nível de protocolo	São feitas implicitamente através de um modelo de transação. O protocolo de mensagens do IIOP, automaticamente propaga as	É necessário que o programador solicite explicitamente, ao dispensador de transações DCOM, por uma referência

	transações CORBA.	de transação, para que esta seja repassada para todas as chamadas subsequentes que o mesmo realize. Isto pode causar mais problemas do que se fosse feito automaticamente.
Referências de Objetos persistentes	Os Objetos CORBA são únicos e possuem referências permanentes.	Os Objetos deste não possuem um estado e não são permanentes.
Nomeação baseada na URL	Um Objeto pode ser associado a uma URL, de forma que ao clicar nesta, uma ligação a um Objeto ativo seja efetuada. Esta característica é muito importante na Internet e nas Intranets.	É suportado através de *monikers*, os quais são utilizados para estabelecer a relação entre os Objetos DCOM e um contexto, uma vez que tais Objetos não possuem um estado.
Invocações de Objetos de multilinguagem	Possui padrões que definem ligações com linguagens de alto nível.	Não há padrões para mapeamento de linguagens para o DCOM. Ele é apenas suportado pelos compiladores da Microsoft.
Padrão aberto	É uma plataforma totalmente aberta, uma vez que não é controlada por uma única empresa e sim por um consórcio de mais de 700.	Embora não pertença mais a Microsoft, ainda sofre muito de suas influências.

Tabela 1 – Diferenças entre CORBA e DCOM.

Tallman et al [19] lembram outros aspectos:

- **Erros e Exceções**

  O Corba possui uma capacidade de exceção que é mapeada naturalmente para linguagens que já possuem tratamento de exceção, tais como C++ e Java, e para aquelas que não o possuem, terá que ser feito um mapeamento em dados de exceção.

  O mecanismo de suporte de erro do Dcom, é feito através de um valor de retorno de 32 bits denominado HRESULT. Já o mecanismo de exceção, é implementado através do transporte do objeto "exceção do objeto" para o cliente. Tal mecanismo

não suporta tipos de exceções definidos pelo usuário. Contudo, o objeto de exceção pode carregar um conjunto fixo de dados fornecidos pelo usuário.

- **Escalabilidade**

  Os defensores do Dcom argumentam que o Corba não é escalável por não implementar o conceito de "*stateless object*" (objetos sem estado). Eles dizem que tais objetos permitem a reutilização de um objeto já instanciado, representando uma série de instâncias, sem atribuir o custo a novas instanciações. Desta forma, o Dcom seria a melhor opção. Ele implementa tal característica através do MTS (*Microsoft Transaction Server*). Contudo, há os que digam que o termo "*stateless object*" não é aplicado corretamente aos objetos do MTS e que este não reutiliza instâncias. Além disso, mesmo o Corba não implementando tal conceito, através de suas implementações já existentes, seus desenvolvedores podem empregá-lo facilmente.

  Atualmente, a escalabilidade de um sistema distribuído depende mais da qualidade do projeto do que do elemento intermediário (*middleware*) propriamente dito. Devido as similaridades destas duas arquiteturas, a análise deve ser feita através da extensão e maturidade dos serviços comuns, pois a disponibilidade destes, terá um impacto direto no projeto. Portanto, neste aspecto o Corba é mais vantajoso que o Dcom, devido aos serviços destes serem relativamente poucos e não testados se comparados aos do Corba.

  Outro aspecto para a escolha do projeto, é o suporte para plataforma. Dependendo do tamanho do sistema, pode ser necessário uma solução com vários tipos de plataformas. O Corba mais uma vez, oferece mais vantagens devido suportar um maior número.

- **Serviços**

  Os serviços do Dcom são integrados com os serviços do sistema operacional. Tal integração permite um ambiente robusto e de crescente flexibilidade para a execução e desenvolvimento de objetos. A Microsoft quer com isso tornar os seus sistemas operacionais a plataforma de escolha para sistemas de baixa a média extensão, enquanto fornece a integração necessária com sistemas de *mainframe*. Os seus serviços são: segurança, gerenciamento do ciclo de vida, tipo de informação, nomeação, acesso ao banco de dados, transferência de dados, registro

e comunicações assíncronas. A comparação deste aspecto entre estas duas arquiteturas não é muito simples, devido a categorização ser feita de formas diferentes. Como exemplo pode-se citar as funções de tipo de informação e registro, que no Dcom são especificadas como serviços e no Corba estão inclusas nas funcionalidades do ORB.

A OMG especificou até o momento quinze serviços, mas não há obrigatoriedade da implementação de todos eles. Os vendedores podem implementar apenas alguns. Os serviços são: nomeação, eventos, ciclo de vida, objetos persistentes, transação, controle de concorrência, relacionamento, externalização, pesquisa, licenciamento, propriedade, tempo, segurança, *trader object* e coleções de objetos. Todos os serviços, com exceção do de persistência, têm se mostrado robustos e úteis nas várias implementações dos diversos vendedores. Onde há ainda pouca experiência na indústria, é em relação aos serviços mais recentes, tais como os de transações.

- **Maturidade**

  Os autores [19] acreditam que a maturidade de uma tecnologia pode ser avaliada pelo número de sistemas em que ela foi empregada. Eles fizeram uma pesquisa e verificaram que em sistemas a nível empresarial, o Corba está sendo muito mais utilizado que o Dcom. Contudo, apesar de não terem encontrado muitas referências sobre projetos do Dcom, eles acham que este deve estar sendo empregado em sistemas a nível de *desktop* e departamental. A experiência e prática dos desenvolvedores é de extrema importância para o sucesso da tecnologia. Neste ponto, o Corba está a pelo menos dois anos adiantado em relação ao Dcom.

5.1.2. Análise das Diferenças sobre Aplicações Específicas

Até agora, a comparação realizada entre estas duas arquiteturas, tem sido feita analisando-se apenas aspectos isolados. Vejamos então uma análise de aplicações de tarefas críticas e de larga escala, sendo aplicadas em uma Intranet, sobre uma Extranet e através da Internet. Os aspectos a serem analisados serão os de escalabilidade, confiabilidade, segurança e gerenciamento. Para maiores informações consultar [16].

5.1.2.1. SUPORTE PARA APLICAÇÕES DE TAREFAS CRÍTICAS

DCOM

- Escalabilidade: através do MTS, as aplicações podem ser estendidas de acordo com a necessidade. Ele fornece um conjunto de interfaces e bibliotecas do Dcom que possibilitam tal extensão. Ele também provê o gerenciamento automático de sub-processos, o que permite componentes individuais gerar vários sub-processos para suportar a crescente demanda, sem que para isso, os desenvolvedores necessitem escrever o código dos sub-processos.

  O serviço de nome centralizado é um componente fundamental em uma arquitetura escalável. Ele permite que usuários localizem uma aplicação ou objeto, estejam eles remotos ou não. Isto possibilita que a aplicação suporte o crescente número de usuários dispersos geograficamente.

  O Dcom ainda não possui este serviço. A partir do Windows NT 5.0, com a adição da Interface de Serviço de Diretório Ativo (ADSI – *Active Directory Service Interface*), componentes poderão utilizar serviços de nomes já existentes, de forma similar a utilizada no serviço de nome centralizado.

- Confiabilidade: o Dcom desenvolve e mantém aplicações confiáveis através do MTS (*Microsoft Transaction Server*), do MSMQ (*Microsoft Message Queue Server*) e do MCS (*Microsoft Cluster Server*). Eles garantem aspectos de confiabilidade, tais como:
  - Entrega de solicitações e desconexão de operações, mesmo que ocorra uma falha temporária na máquina servidora da aplicação ou na rede;
  - Atualização de dados armazenados em depósitos dispersos, através do uso de transações distribuídas;
  - Reinicialização de aplicações automaticamente e substituição automática de máquinas servidoras em caso de falha.

  O MTS está na sua 2ª versão, enquanto que os demais ainda precisam ser mais testados.

- Segurança: existem várias maneiras de se implementar este aspecto, cada uma fornecendo um tipo de serviço, os quais não podem ser integrados, isto é, não trabalham conjuntamente em uma mesma solução.

- Gerenciamento: o MMC (*Mirosoft Management Console* – Unidade de Gerenciamento da Microsoft) fornece uma interface gráfica de usuário unificada (GUI – *Graphical User Interface*), para o gerenciamento de componentes básicos do MTS e do MSMQ. Os aspectos gerenciados são a administração e configuração centralizada, e o emprego remoto de componentes. O MMC é uma ferramenta recente e sua utilidade a nível departamental, ainda não é totalmente segura.

Apesar de todas estas ferramentas e tecnologias oferecidas pelo Dcom para o desenvolvimento das soluções, já apresentadas, para o suporte de aplicações de tarefas críticas, existem ainda dois importantes obstáculos. Um deles é que apesar do Dcom poder ser executado em uma plataforma UNIX através do software AG, ele é uma solução basicamente para o Windows. A tecnologia disponível para o UNIX está a pelo menos uma revisão atrasada em relação ao Windows. Além de que ferramentas importantes como o MTS, MSMQ e *Microsoft Message Queue,* não estão disponíveis para ele também. O outro problema é que muitas das ferramentas utilizadas são muito recentes e ainda estão amadurecendo a nível empresarial.

CORBA

- Escalabilidade: possibilita o uso eficiente de recursos, pois os objetos são instanciados apenas quando solicitados. A independência da localização das aplicações e dos usuários, é fornecida pelo serviço de Nome centralizado. E o serviço de *Trader* oferece um modelo de procura de componentes como os das "Páginas Amarelas". O balanceamento de carga estático entre várias réplicas de uma aplicação é disponibilizado pelo serviço de Nome.

- Confiabilidade: é responsabilidade do serviço de transação. Em alguns vendedores, a implementação deste serviço é recente. Já em outros, estas implementações são baseadas nos monitores de processamento de transação, já experimentados nas indústrias. Algumas extensões proprietárias do Corba, fornecem suporte em caso de falhas tanto para as aplicações servidoras quanto às plataformas servidoras.

- Segurança: existem dois níveis de segurança definidos pelo Corba. O primeiro, é para as aplicações que não têm conhecimento da segurança, mas que são executadas em um

domínio seguro. Há a autenticação do usuário, autorização através de listas de controle, encriptação e integridade dos dados, e não-repudiação, que é um item opcional.

Já no outro nível, as aplicações sabem sobre a segurança e por isso os itens citados acima, precisam estar em uma versão mais robusta. Ainda não há nenhuma implementação deste nível.

- Gerenciamento: existem ferramentas sofisticadas que permitem a configuração e administração centralizada de aplicações Corba. A Iona permite também o gerenciamento centralizado a partir de qualquer SNMP (*Proprietary Management Network System* – Sistema de Rede de Gerenciamento Proprietário), que é concorrente do MMC.

O Corba permite que os desenvolvedores optem pela melhor implementação dos serviços, dos diversos vendedores. Contudo, fazer com que uma implementação de segurança de um vendedor interaja com o serviço de transação de um outro vendedor, é praticamente impossível. E como nenhum vendedor tem implementado todas as soluções necessárias ao suporte de aplicações de tarefas críticas já mencionadas, esta é um questão que surge sempre que o Corba é utilizado para criar tais aplicações.

5.1.2.2. SUPORTE PARA APLICAÇÕES *WEB*

Primeiramente deve ser esclarecido o que é uma aplicação *Web*. Uma aplicação *Web* não é meramente uma aplicação disponibilizada através da Internet, e sim, uma aplicação em que a sua estrutura de apresentação é um navegador de *Web*. Aplicações em redes locais e em *intranets* com interfaces de navegadores, estão sendo muito desenvolvidas devido possuírem uma interface universal e de fácil disponibilização.

Aplicações *Web* devem ser possíveis de ser executadas tanto no *Internet Explorer* quanto no *Netscape Navigator*, em todas as plataformas. Além disso, elas devem ser fáceis e rápidas de carregar do servidor *Web*, sem que o cliente necessite instalar algo mais. A aplicação também deve poder ser integrada com componentes de outros navegadores, tais como formas e *frames* (quadros).

DCOM

As aplicações *Web* que são desenvolvidas pelo Dcom com a interface na forma do controle do ActiveX, podem ser executadas no *Internet Explorer* e, através de um *plug-in,* no *Netscape Navigator*. Os controles do ActiveX são plataformas binárias específicas e são necessários conjuntos binários diferentes para cada plataforma. Mesmo assim eles funcionam tanto em plataformas Unix quanto Windows. Apesar destes controles estarem disponíveis originalmente nas plataformas Wintel, independente do navegador utilizado, a aplicação não necessita carregar os serviços executáveis.

Através da tecnologia do *Microsoft Active Server Page*, clientes HTML e ActiveX, podem ser integrados da mesma forma com os servidores de aplicação Dcom. Esta tecnologia também possibilita que serviços como o MTS e MSMQ sejam utilizados pelas aplicações *Web*.

CORBA

Devido muitos vendedores estarem desenvolvendo implementações de serviços e executáveis do Corba totalmente em Java, interfaces baseadas em Java para aplicações Corba, podem ser executadas na maioria dos navegadores e plataformas.

Interfaces disponíveis através do Netscape podem ser carregadas mais rapidamente, uma vez que elas não precisam carregar o executável do Corba.

Os serviços Corba são disponibilizados tanto como arquivos comprimidos Java, quanto como arquivos Microsoft para *downloads* do servidor Web mais rápidos.

Clientes baseados em HTTP podem se comunicar com servidores de aplicações Corba através da Interface de Aplicação Web, que é fornecida pelo *Nestcape Enterprise Server*.

5.1.2.3. SUPORTE PARA INTERNET E EXTRANET

Aplicações de tarefas críticas empregadas através da Internet ou de uma extranet possuem algumas características básicas:

- Geralmente atuam em longas distâncias e através de muitos *firewalls*;
- Podem possuir uma interface Web ou uma GUI (*Guide User Interface* – Interface Guia do Usuário) cliente/servidor.

Estas aplicações têm dois requisitos muito importantes, que são os de segurança e o suporte a *firewall*. Transações de tarefas críticas que são executadas através da Internet, necessitam de medidas de segurança adicionais para garantir a precisão, confidencialidade e credibilidade.

DCOM

Atualmente o Dcom provê dois fatores de autenticação e suporte ao Serviço de Dados Remotos (RDS) para aplicações na Internet e em extranets. Esses fatores são suportados através de certificados públicos e cartões inteligentes. O suporte para o SSL (*Secure Socket Layer* – Camada de Segurança de Socket) e sua integração com a segurança do STLM ainda está limitado.

CORBA

Muitos dos vendedores do Corba possuem suporte ao SSL (*Secure Socket Layer* – Camada de Segurança de Socket) e tem também os dois fatores de autenticação. Contudo, ambos ainda são muito imaturos. Alguns dos vendedores possibilitam a integração das suas soluções de segurança com outras já existentes como DCE e Kerberos. A maioria das soluções de segurança implementadas pelo Corba, são de natureza proprietária. Contudo, elas estão sendo padronizadas para a especificação da versão 3.0 do Corba.

5.1.2.4. SUPORTE PARA APLICAÇÕES INTRANET

Aplicações intranet que são utilizadas dentro de uma organização, devem ser otimizadas, não importando a largura de banda da rede e se não tem restrições à *firewall*. Esta otimização pode ser através das interfaces do usuário e das funcionalidades.

As interfaces do usuário devem possuir um bom suporte a gráficos e também devem ser capazes de integrar a aplicação com ferramentas de *desktop* populares, tais como *Microsoft Office* e *Visual Basic*.

E nas funcionalidades, devem estar incluídos requisitos de persistência e *caching* de objetos locais e a capacidade de divulgação e assinatura.

DCOM

O Dcom possibilita o desenvolvimento de aplicações intranet otimizadas, pois ele cumpre todos os requisitos citados acima.

- A maioria dos ambientes de desenvolvimento das plataformas Wintel, suportam o rápido desenvolvimento de aplicações gráficas Dcom;
- As ferramentas de *desktop* populares suportam originalmente o ActiveX, facilitando a integração das aplicações Dcom com estas ferramentas;
- OLE DB juntamente com seus Objetos de Dados Ativos, viabilizam o suporte à persistência por parte das aplicações Dcom;
- MSMQ fornece as capacidades de publicação e assinatura.

CORBA

O Corba também viabiliza a criação de aplicações otimizadas, sendo que suas aplicações têm que ser integradas com as ferramentas de *desktop* através de uma ponte COM/Corba, disponibilizada por vários de seus vendedores.

Para os requisitos de persistência, publicação e assinatura, podem ser utilizados os serviços Corba de Eventos e Persistência.

5.2. Uma Avaliação na Integração do Dcom e do Corba para o Suporte de Aplicações do Mundo-Real

Nenhuma das plataformas oferece uma solução completa para a maioria das aplicações realmente implementadas. Geralmente essas aplicações mesclam aspectos, como por exemplo, uma interface Web em uma aplicação intranet ou Internet. Portanto, a melhor solução é a integração das duas tecnologias.

Vejamos como poderia ser esta integração em alguns casos, de forma a melhorar a peformance das aplicações [16].

- APLICAÇÕES INTRANET

A interface do usuário deve ser criada através do ActiveX/Dcom. Isto possibilita à aplicação acesso às melhores capacidades do Dcom na intranet. Além da integração com as ferramentas de *desktop* populares.

Posteriormente esta interface deve ser integrada aos servidores de aplicação Corba via uma ponte COM/Corba. Estes servidores permitem que a aplicação seja executada através de plataformas heterogêneas, viabilizando a sua escalabilidade através do serviço de Nome Centralizado do Corba.

- APLICAÇÕES INTERNET/EXTRANET BASEADAS EM WEB

Nestas aplicações, as interfaces do usuário devem ser baseadas no Corba, as quais são *applets* escritas totalmente em Java. A comunicação é feita através de um tunelamento do HTTP para o IIOP com os servidores de aplicação Dcom, empregados geralmente no MTS.

Desta forma, o requisito principal das aplicações Web, que é permitir que as *applets* possam ser executadas em qualquer navegador de qualquer plataforma, é satisfeito. Além disso, as *applets* também poderão utilizar a segurança do SSL e o túnel HTTP, para acessar servidores de aplicação através de *firewalls*, satisfazendo desta vez requisitos de Internet. Ao utilizar os servidores Dcom no *Microsoft Transaction Server*, a aplicação por conseguinte, também utiliza o suporte de gerência, transação e sub-processos do MTS.

- APLICAÇÕES DE TAREFAS CRÍTICAS

Neste caso, a decisão de integrar, como integrar, ou mesmo de não integrar estas duas tecnologias, depende da forma como se deseja empregar estas aplicações e da importância dos requisitos de cada aplicação para a empresa em questão.

Ambas as tecnologias provêem uma infra-estrutura robusta, sendo que suas soluções não são completas para todos os tipos de aplicações de tarefas críticas.

5.3. SEMELHANÇAS ENTRE DCOM E CORBA

Contudo, não existem apenas diferenças entres estas duas arquiteturas, há também semelhanças. Dentre estas estão alguns fatores tais como:

- Nível de Abstração – ambas possuem um ótimo nível;
- Suporte para tipos de parâmetros – fornecem todos os tipos de parâmetros, entrada, saída e entrada/saída;
- Facilidade de configuração - possuem certo nível de facilidade, porém não o mais desejado, uma vez que não é tão simples configurar sistemas Cliente/Servidor;
- Invocações Dinâmicas - suportam tal tipo de invocação, permitindo a construção de um método em tempo de execução.

Existem ainda muitos outros fatores que devem ser analisados no momento da escolha de uma plataforma deste tipo. É necessário analisar o objetivo para o qual se pretende utilizar tal tipo de solução, o tamanho da organização onde será implantada a plataforma. Somente então, a decisão de qual se adequará melhor ao propósito requerido, poderá ser feita.

5.4. OPINIÕES

Diversas e divergentes são as opiniões sobre este tema. Há os defensores do Dcom, assim como há os do Corba. Mas existem também aqueles que concordam que, dependendo dos requisitos da aplicação, a melhor plataforma para aplicações distribuídas orientadas a Objetos, é uma combinação destas duas arquiteturas.

Segundo Rosemary Rock-Evans [20], "o Corba é uma tecnologia que não tem possibilidade alguma de se igualar ao Dcom". A entrevistada acredita que devido os fornecedores de monitores de processamento de transações distribuídas (DTPM – *Distributed Transactions Processing Monitors*), estarem começando a suportar a tecnologia de objetos, não haverá mais a necessidade de se utilizar um ORB, uma vez que um DTPM oferece confiabilidade, disponibilidade, larga aplicabilidade e segurança, tudo com a habilidade do uso de objetos.

Ainda seguindo o ponto de vista de Rock-Evans, o problema com os ORBs do Corba, é que todo o serviço, por mais simples que seja, deve ser invocado pelo usuário através de

vários comandos. "Alguns ORBs possuem 700 comandos – isto torna o desenvolvimento muito difícil quando você tem 700 comandos para aprender".

Outro problema fundamental que Rosemary Rock-Evans aponta, é que nenhum dos ORBs baseados no Corba trabalham juntos. Ela acredita que no mercado dos ORBs, o principal vencedor será a Microsoft, seguido pela IONA e Visigenic.

Na [26], John Montgomery comenta sobre uma lógica tradicional que diz algo como: "Utilize o Corba quando você precisar interconectar com uma plataforma que não seja Windows; e o Dcom apenas para sistemas Windows". Acontece, ele continua, que a Microsoft transferiu o gerenciamento do Dcom para o *Open Group* e introduziu o *Transaction Server* (Servidor de Transação) como parte da sua Plataforma Ativa. Existe ainda o Software AG que está tornando o Dcom possível de ser executado em outras plataformas além do Windows. Portanto, o que já foi de completo domínio do Corba, ambientes heterogêneos em que o mesmo programa pode ser executado de forma idêntica, agora está ficando sob o domínio do Dcom também.

Quanto ao Corba, o Netscape ao incorporar ao seu navegador um ORB IIOP e devido também a grande popularidade do Java, fez com que surgisse uma solução de *desktop* para o mesmo, o que antes era apresentado apenas pelo COM.

Dito isto, o autor conclui que vendedores de softwares independentes (ISVs – *Independent Software Vendors*) e corporações de desenvolvedores de software, devem reconsiderar suas estratégias de componentes distribuídos. Desta forma, surgiriam ambientes utilizando ambas as tecnologias, interligando-as através de *gateways*.

VI CAPÍTULO

IMPLEMENTANDO UM APLICATIVO DE BANCO DE DADOS DISTRIBUÍDOS EM CORBA E DCOM

No capítulo anterior foi realizada uma comparação entre as tecnologias Corba e Dcom, analisando-se vários aspectos a fim de determinar as características de cada uma delas em relação aos mesmos. Seus comportamentos foram avaliados a partir da observação de aspectos isolados e considerando-se também um ambiente de aplicação.

Contudo, esta análise baseou-se em documentos consultados, que foram redigidos por outros autores, ou seja, foi um estudo teórico. Tão importante quanto possuir uma base teórica, é obter também uma experiência prática do que se está estudando, adquirindo assim conclusões próprias.

Foi visando tal objetivo que idealizou-se uma aplicação dentro de um determinado ambiente, que deveria ser implementada tanto na tecnologia Corba quanto Dcom, a fim de observar o comportamento de cada uma considerando aspectos como: nível de facilidade de implementação, material de consulta e auxílio disponível, recursos necessários para sua execução e o aspecto de portabilidade. Este capítulo descreve as etapas presentes no desenvolvimento de tal aplicação.

6.1. DESCRIÇÃO DA APLICAÇÃO

A aplicação desenvolvida, é uma aplicação de banco de dados distribuídos, que foi implementada no ambiente Delphi 4, na versão Cliente/Servidor. Trata-se do banco de dados de um hotel, o **PitAnita's Hotel**. O programa utilizado para escrever e armazenar as tabelas do banco de dados, foi o *Database Desktop*. Este é instalado juntamente com o Delphi.

A princípio, o objetivo era utilizar o Visual J++ 6.0, pois pretendia-se realizar um teste com um aplicativo já existentes e disponibilizado na Internet. Este teste serviria como base para a criação de uma nova aplicação. Contudo, não foi possível obter tal programa para instalação.

Outro programa cogitado foi o JBuilder 3.0 por já existir em determinada máquina do Laboratório de Sistemas Distribuídos, ao qual tenho acesso. Mais uma vez encontrou-se um obstáculo que tornou inviável sua utilização. Tal obstáculo foi a dificuldade em encontrar livros e materiais afins que auxiliassem no seu aprendizado.

Por fim optou-se pelo Delphi 4, pois havia grande facilidade em conseguí-lo para instalação e também na obtenção de livros e *sites* de ajuda. Além de ser um ambiente que já traz recursos que facilitam a integração de clientes e servidores através das plataformas Corba e Dcom.

Abaixo está a figura 6.1 que descreve de forma genérica a estrutura da aplicação.

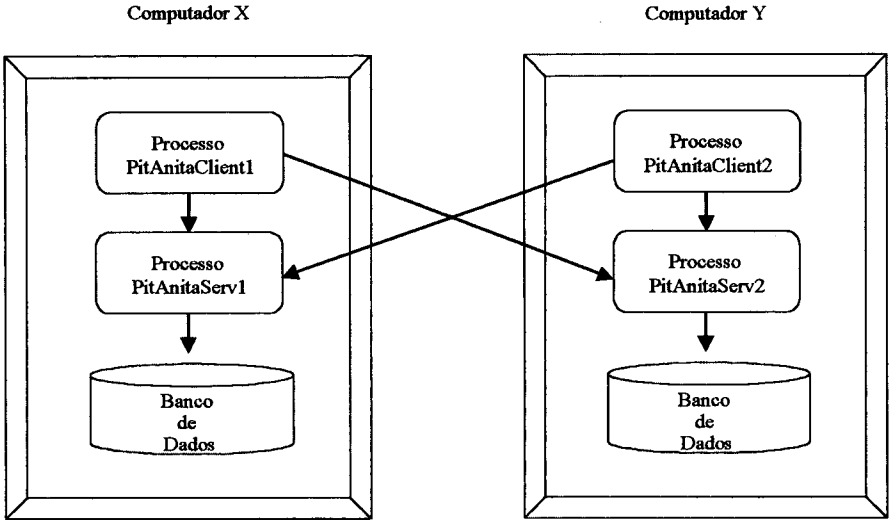

Figura 6.1 – Diagrama da Estrutura da Aplicação do PitAnita's Hotel

A opção de utilizar o Delphi 4 como o ambiente para o desenvolvimento dos clientes e servidores, se deu depois de um levantamento dos ambientes orientados a objetos existentes. Embora este não tenha sido o primeiro ambiente escolhido, devido a alguns problemas como conseguir o programa para instalação e materiais (*sites* e livros) para o auxílio de seu manuseio, este foi o que melhor se adaptou às condições requeridas.

Uma vez que se trata de uma aplicação distribuída, haverão clientes e servidores se comunicando, trocando informações. Os servidores são aqueles que terão acesso as tabelas. Os clientes as acessarão através dos mesmos via solicitações. Eles podem solicitar desde simples consultas até a alterações dos dados.

6.2. DESCRIÇÃO DA IMPLEMENTAÇÃO

6.2.1. *ALIAS*

Um *alias* (apelido) é um conjunto de parâmetros que descreve uma conexão de banco de dados. Ele é um nome criado para especificar o diretório em que está armazenado o banco

de dados e, no caso de uma aplicação cliente/servidora remota, também contém outras informações adicionais. Na aplicação desenvolvida, foi utilizado o BDE (*Borland Database Engine) Administrator* para a sua criação.

O BDE é uma camada de software que cria uma interface uniforme entre aplicações Delphi e os bancos de dados propriamente ditos. Ele fornece a característica de transparência em relação ao banco de dados utilizado, uma vez que o programador Delphi, apenas necessita ter conhecimento do *alias* do BD no sistema. Em caso de uma mudança, por exemplo, do tipo de plataforma de BD utilizada, apenas seu *alias* precisará ser alterado.

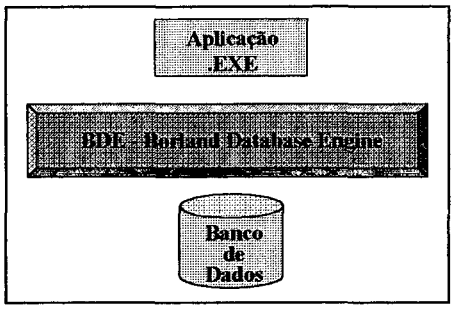

Figura 6.2 – BDE (*Borland Database Engine*)

O componente *Table* utilizado pelo servidor para se conectar com a tabela desejada, possui uma propriedade denominada *Database Name*, através da qual o nome do banco de dados ao qual a tabela pertence é especificado. E este nome é justamente o *alias*.

6.2.2. TABELAS

A forma escolhida para o armazenamento dos dados foi através de tabelas e, após uma análise dos objetivos da aplicação, verificou-se a necessidade da criação de quatro tabelas. O programa de banco de dados utilizado foi o *Database Desktop* e o tipo de tabela criada foi o *Paradox* 7. Este número corresponde a versão do programa. As quatro tabelas criadas foram as seguintes:

a) Tabela de Lotação

Esta tabela lista todos os quartos do hotel, se estão ou não ocupados, para quantas pessoas é a sua capacidade e o seu preço.

Nº Quarto	Disponibilidade	Capacidade	Preço

- Nº Quarto: é tido como chave primária desta tabela;
- Disponibilidade: pode receber o valor "livre" ou "ocupado", indicando a situação do quarto;
- Capacidade: indica para quantas pessoas é o quarto. Se é duplo, triplo ou para até quatro pessoas;
- Preço: indica o valor de cada quarto de acordo com a sua capacidade.

b) Tabela de Ocupação

Lista todos os hóspedes que estão hospedados no hotel no momento, o número do quarto em que estão, a data em que chegaram e a que pretendem sair.

CPF Hóspede	Nome Hóspede	Nº Quarto	Dt Entrada	Dt Prev Saída

- CPF Hóspede: número do cadastro de pessoa física do cliente responsável. É uma chave primária;
- Nome Hóspede: nome do hóspede responsável;
- Nº Quarto: também é uma chave primária. A busca pode ser feita tanto por este campo, como pelo CPF do hóspede;
- Dt Entrada: é a data em que o hóspede é registrado no hotel;
- Dt Prev Saída: é a data de previsão de saída fornecida pelo hóspede, a fim de que o hotel possa ter um controle para as demais reservas.

c) Tabela de Reservas

É a tabela que contém futuras reservas. É necessária para o controle das vagas disponíveis. Além do CPF e nome do hóspede responsável pela reserva, também contém para quantas pessoas é a mesma, a data da chegada e o tempo que deverão permanecer.

CPF Hóspede	Nome Hóspede	Total Pessoas	Dt Reserva	Duração

- CPF Hóspede: número do cadastro de pessoa física de quem efetua a reserva. É uma chave primária;
- Nome Hóspede: nome do hóspede responsável;
- Total Pessoas: para quantas pessoas é a reserva;
- Dt Reserva: a data da futura chegada e registro do hóspede;
- Duração: o tempo pretendido pelo hóspede para permanecer no hotel.

d) Tabela de Cadastro de Hóspedes

Esta tabela contém o registro pessoal de todos os hóspedes, para o caso de haver necessidade de um futuro contato ou mesmo de uma confirmação pelas mais diversas razões.

CPF Hósp	Nome Hósp	Endereço	Fone	Cidade	Estado	País	Período Estadia

- CPF Hósp: número do cadastro de pessoa física do hóspede responsável. É uma chave primária;
- Nome Hósp: nome do hóspede responsável;
- Endereço, Fone, Cidade, Estado, País: dados pessoais do hóspede;
- Período Estadia: mês e ano de sua hospedagem.

Foi criada uma relação entre as tabelas de Ocupação e Lotação, uma vez que um registro da primeira só poderá existir caso o número do quarto em questão, esteja registrado na segunda. Também há uma relação entre as tabelas de Ocupação e Cadastro de Hóspedes, pois esta armazena dados de hóspedes que já estiveram no hotel, o que obrigatoriamente significa que já estiveram registrados na tabela de Ocupação.

Após a criação das tabelas, ainda no mesmo programa, lhes foram acrescentados alguns dados, de forma que elas não estivessem vazias no momento em que fossem acessadas pelo cliente no Delphi através do servidor.

6.2.3. SERVIDOR CORBA

Tanto o servidor quanto o cliente foram implementados utilizando-se o ambiente orientado a objetos Delphi 4. Esta versão deste ambiente, já traz ferramentas que facilitam a interação entre clientes e servidores Corba.

O primeiro que deve ser criado naturalmente é o servidor. Ele conterá os componentes que terão acesso às tabelas. No caso desta aplicação, foram criados dois servidores. Um deles contendo as tabelas de Lotação e Ocupação, e o outro, as de Reserva e Cadastro de Hóspedes. Na verdade, bastaria que fosse criado um único servidor contendo todas estas tabelas. Contudo, com o intuito de prevenir falhas em caso de haver algum problema com um destes servidores, e também para a demonstração do uso de mais de um servidor por um mesmo cliente simultaneamente, concluiu-se que seria mais conveniente a repartição das tabelas em mais de um servidor.

Os dois servidores criados foram o PitAnitaCorbaServ1 e o PitAnitaCorbaServ2. Ambos contém um Módulo de Dados Remoto Corba, com duas tabelas específicas cada. Estes módulos são as interfaces dos objetos já mencionados em capítulos anteriores, que deverão ser adicionadas ao repositório de interface, de forma a disponibilizá-las aos clientes.

No momento da criação do módulo de dados, deverá ser feita a opção pelo tipo de instância e o modelo de *threading* (fluxo de controle) que será adotado. Também deverá ser fornecido o nome da classe deste objeto. Este é um nome aleatório escolhido pelo programador.

As opções para o tipo de instância são:

- *Instance-per-client* (instância por cliente): para cada conexão do cliente, será criada uma nova instância do objeto, que persistirá enquanto a conexão durar. Uma vez que clientes separados não interferem nas configurações das propriedades uns dos outros, é possível utilizar informações com estado persistente;
- *Shared instance* (instância compartilhada): uma única instância do objeto suporta as solicitações de todos os clientes. Devido a instância ser compartilhada por todos os clientes, informações com estado permanente não são confiáveis, tal como configuração de propriedades.

Os modelos de fluxos de controle disponíveis são:

- *Single-threaded* (Único Fluxo de Controle): cada instância de objeto recebe apenas uma solicitação por vez;

- *Multi-threaded* (Vários Fluxos de Controle): cada conexão do cliente possui seu próprio fluxo de controle, o que permite à aplicação ser chamada por vários clientes simultaneamente. Escrever servidores *multi-threaded* quando se está utilizando instâncias de objetos compartilhadas pode ser perigoso, pois há a necessidade de proteger todos os dados e objetos do servidor, contra conflitos entre esses fluxos.

A figura 6.3 exibe o módulo de dados do Corba, onde deverão ser especificadas as características acima descritas. Em particular, os dados que aí estão são para a criação do módulo de dados da tabela de Lotação contida no servidor PitAnitaCorbaServ1.

Como a aplicação aqui desenvolvida é uma aplicação bastante simples, envolvendo apenas dois clientes, optou-se por utilizar o modelo de instância que cria um instância do objeto para cada cliente, pois não acarretaria em um grande desperdício de memória; e pelo modelo de vários fluxos de controle, uma vez que não haveria problemas de conflito entre os fluxos já que existe uma instância para cada cliente.

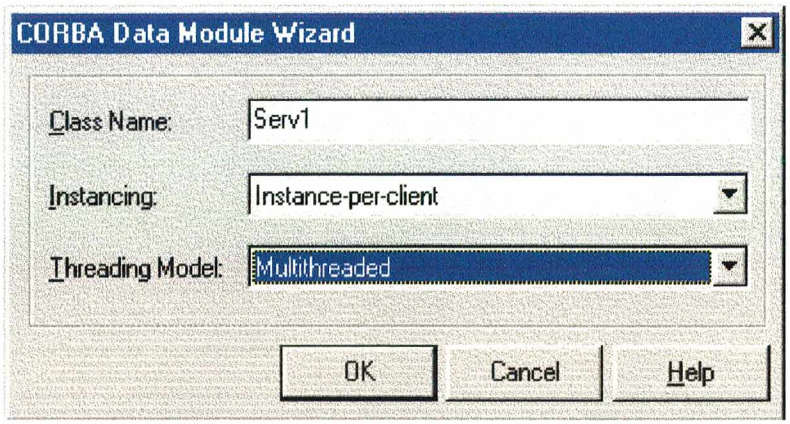

Figura 6.3 – Criando um Módulo de Dados Corba

O próximo passo é acrescentar ao módulo tantos componentes *TTable* quanto forem o número de tabelas requeridas, no caso duas, conectar este componente ao BD e tabela requeridos, ativá-lo e exportá-lo de forma que sua tabela possa estar visível aos usuários.

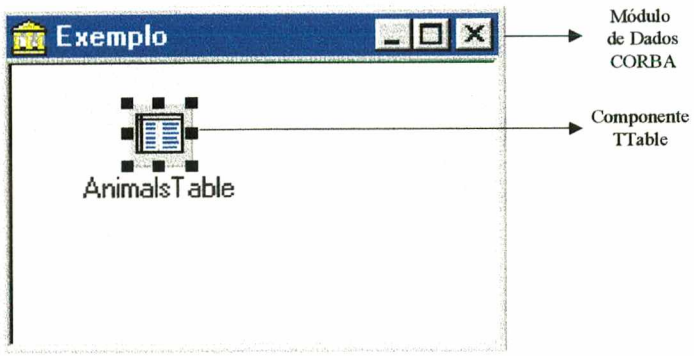

Figura 6.4 – Exemplo da Criação do Servidor Corba

A biblioteca de tipos deve ser aberta e visualizada, para que se possa converter as interfaces dos objetos geradas na IDL do Corba e dessa forma adicionar o módulo de dados ao repositório de interfaces. No Delphi 4, o mapeamento do Pascal para o Corba IDL é feito de forma indireta, pois tem que passar pelo COM, isto é, o Delphi primeiramente transforma do Pascal para a IDL do COM, para somente então converter para a IDL do Corba.

6.2.4. CLIENTE CORBA

A aplicação cliente é conectada ao servidor através de um componente MIDAS do Delphi, denominado *CorbaConnection*.

Este possui algumas propriedades que deverão ser preenchidas, tal como o nome do *host*, que é o nome do computador em que o servidor está localizado; o identificador do repositório, que é o nome do servidor mais o da interface do objeto que foi adicionada ao repositório de interfaces, que é o nome da classe do módulo de dados; e o nome do objeto, que é o nome da instância do objeto, o qual também é o nome do módulo. No caso de uma aplicação cliente/servidora local, não é necessário fornecer nem o nome do objeto e nem o do *host*.

Feito isto, deverá ser acrescentado um componente *ClientDataSet* e um *DataSource*. No momento em que o *ClientDataSet* é ativado, se o servidor estiver rodando e o ORB estiver ligado, o cliente já estará conectado a ele. Para visualizar então os dados, basta acrescentar os campos da tabela remota.

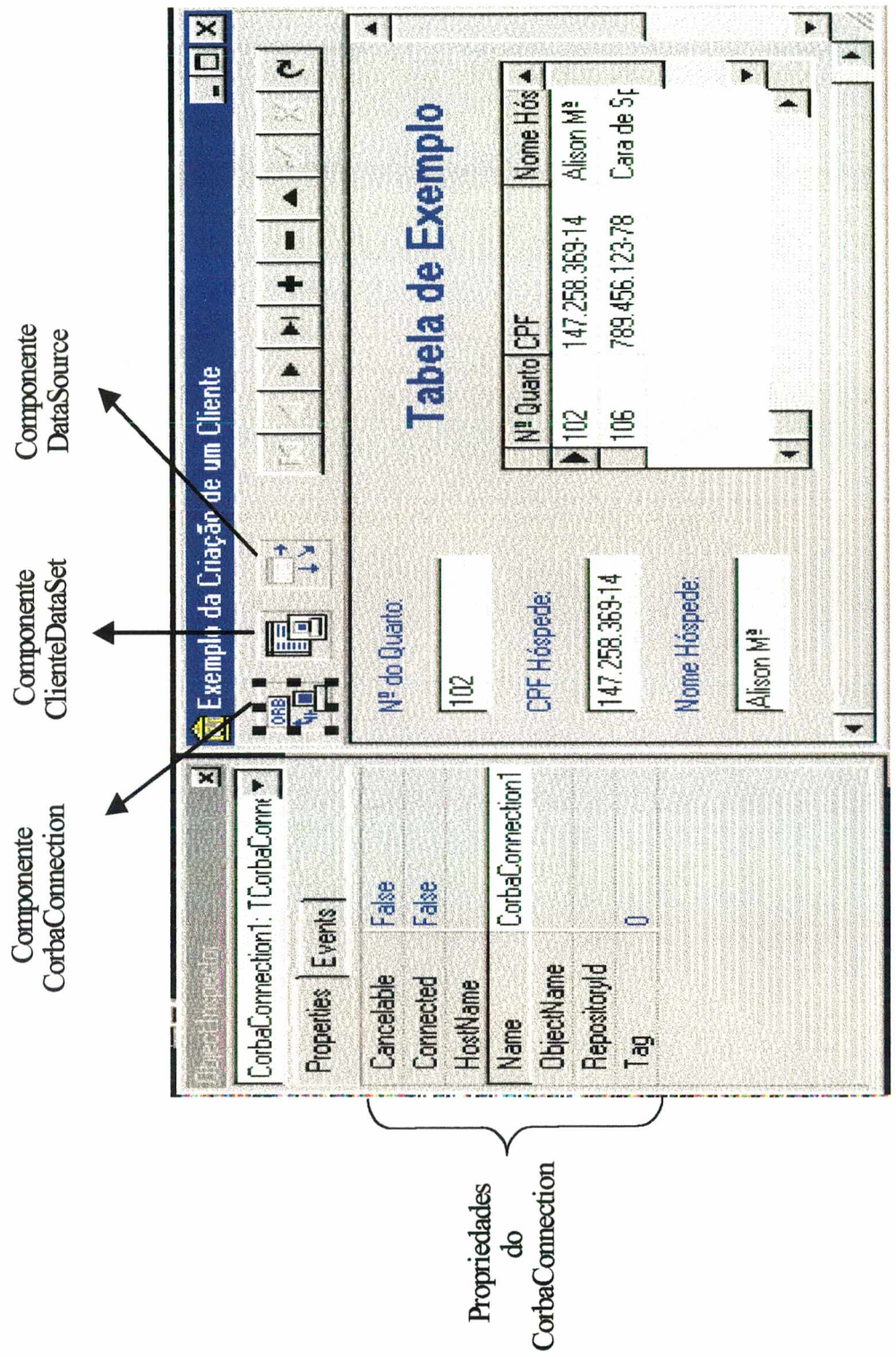

Figura 6.5 – Exemplo da Criação de um Cliente

Na aplicação aqui desenvolvida existem dois clientes que estão localizados tanto local quanto remotamente em relação aos servidores. O PitAnitaCorbaClient1 está local para o PitAnitaCorbaServ1, e remoto para o PitAnitaCorbaServ2. E o PitAnitaCorbaClient2, está local para o PitAnitaCorbaServ2 e remoto para o PitAnitaCorbaServ1. Esta aplicação foi assim desenvolvida para que ambas situações pudessem ser demonstradas, expostas e testadas.

O PitAnitaCorbaClient1 trata dos funcionários do próprio *PitAnita's Hotel*, que efetuarão registros e baixas de hóspedes, cadastro pessoal destes hóspedes e reservas realizadas diretamente ou via outra empresa de turismo que não a *PitAnita's Tur*, que é o PitAnitaCorbaClient2. Esta é uma agência filiada ao hotel em questão, que possui livre acesso para verificar a situação de lotação do hotel e efetuar reservas.

A seguir é feita uma descrição das tentativas realizadas para a implementação desses clientes e servidores.

Primeiramente foram idealizados um único servidor e cliente, localizados em uma mesma máquina. Tal máquina era um *Pentium*100 de 16MB de memória, com o *Windows* 95 como Sistema Operacional.

Após esta primeira fase que foi ultrapassada com sucesso, iniciou-se a tentativa para a implementação remota ainda com apenas um cliente e um servidor. A outra máquina utilizada também possuía o *Windows* 95, mas era de uma outra sub-rede. A tentativa fracassou. Cogitou-se então do problema estar relacionado com a questão das sub-redes diferentes. Tentou-se então uma outra máquina da mesma sub-rede. Acontece que as outras únicas duas máquinas da mesma sub-rede que tinham capacidade para rodar o Delphi, possuíam o *Windows* 98. De qualquer forma, novamente a tentativa fracassou.

Realizou-se em seguida um teste com duas máquinas da Rede Acme, que rodavam o *Windows* 98. Funcionou. Tentei então executar o servidor em uma dessas máquinas e na Sky do LSD. Não obtive sucesso. Porém com outra máquina da mesma Sub-Rede que possuía o *Windows* 98, funcionou.

Para ilustrar melhor o que acabou de ser explanado, observe a figura e tabela abaixo.

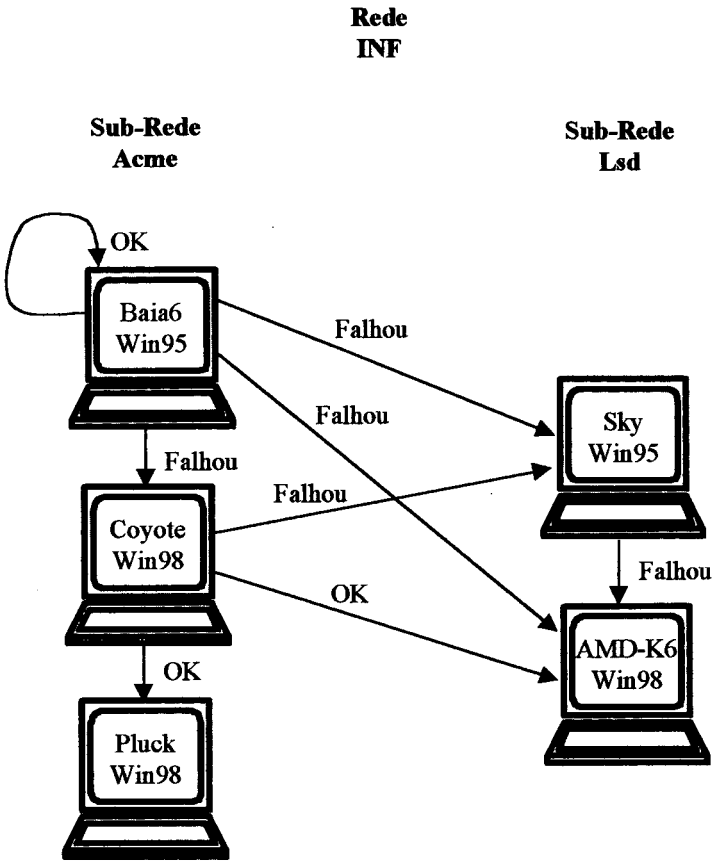

Figura 6.6 – Demonstração das Tentativas para Implementação do Cliente e Servidor Corba

Máquina / Sub-Rede / SO	Máquina / Sub-Rede / SO	Resultado Obtido
Baia6 / Acme / Win 95	Baia6 / Acme / Win 95	Implementação local → Ok
Baia6 / Acme / Win 95	Sky / LSD / Win 95	Implementação remota → Falhou
Baia6 / Acme / Win 95	Coyote / Acme / Win 98	Implementação remota → Falhou
Baia6 / Acme / Win 95	AMD-K6 / LSD / Win 98	Implementação remota → Falhou
Coyote / Acme / Win 98	Pluck / Acme / Win 98	Implementação remota → Ok
Coyote / Acme / Win 98	Sky / LSD / Win 95	Implementação remota → Falhou
Coyote / Acme / Win 98	AMD-K6 / LSD / Win 98	Implementação remota → Ok

Tabela 6.1 – Resumo das Tentativas para a Implementação do Cliente e Servidor Corba

Detectou-se enfim, que o problema estava na utilização do *Windows* 95 para a implementação remota. Consultei um grupo de discussão para tentar saber se havia alguma

restrição referente a tal implementação, mas a resposta obtida foi negativa. E eles também não souberam dizer o que poderia estar ocorrendo, ficando o problema assim sem solução.

O aplicativo final é composto por dois servidores e dois clientes situados em duas máquinas distintas. Tais máquinas pertencem a sub-redes diferentes e ambas rodam o *Windows* 98. Os dois clientes acessam os dois servidores.

6.2.5. RODANDO A APLICAÇÃO

Para executar uma aplicação Corba desenvolvida no Delphi, é necessário primeiramente abrir o ORB na máquina servidora, que no caso é o OSAGENT da VisiBroker que já vem com esta versão do Delphi. Posteriormente, o servidor deverá ser executado. Isto deve ser em um outro depurador que não o Delphi, no caso do servidor se encontrar na mesma máquina que o cliente. Quando o servidor já estiver rodando, então é a vez do cliente. Basta executá-lo no próprio Delphi.

No caso da nossa aplicação, dois ORBs deverão ser abertos, um em cada máquina servidora, depois os dois servidores deverão ser executados ou através do DOS ou do Explorando do Windows, e somente então os clientes.

Abaixo estão as figuras 6.7, 6.8 e 6.9 que mostram, respectivamente, a janela do ORB que é aberta, um servidor e um cliente em execução.

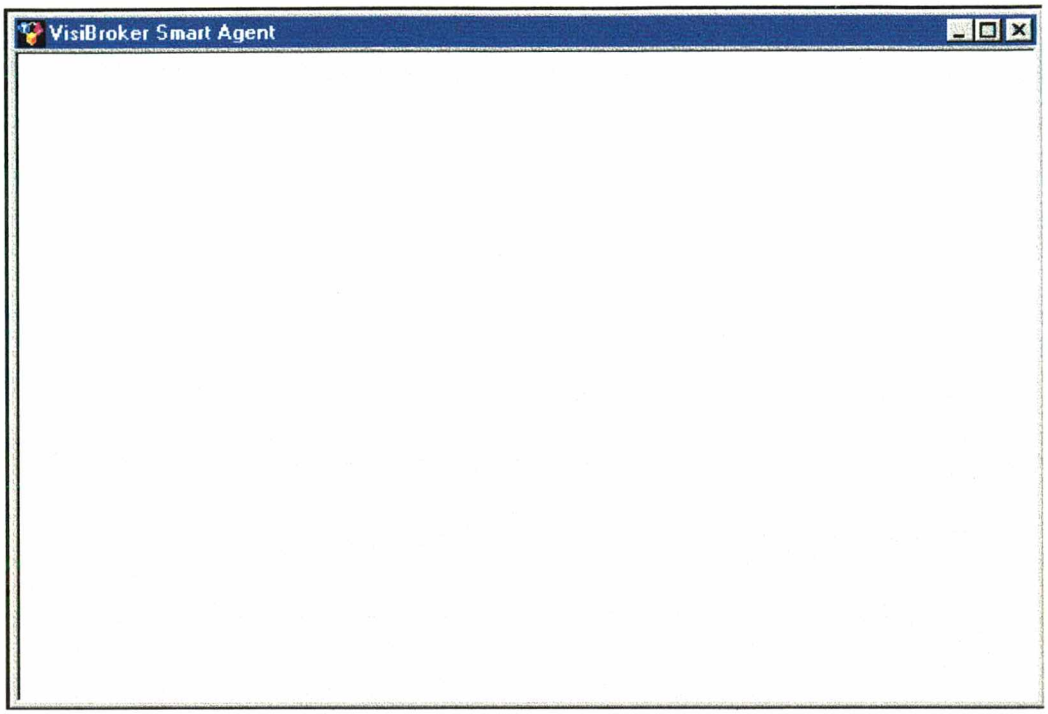

Figura 6.7 – "VisiBroker Smart Agent" : o ORB da VisiBroker

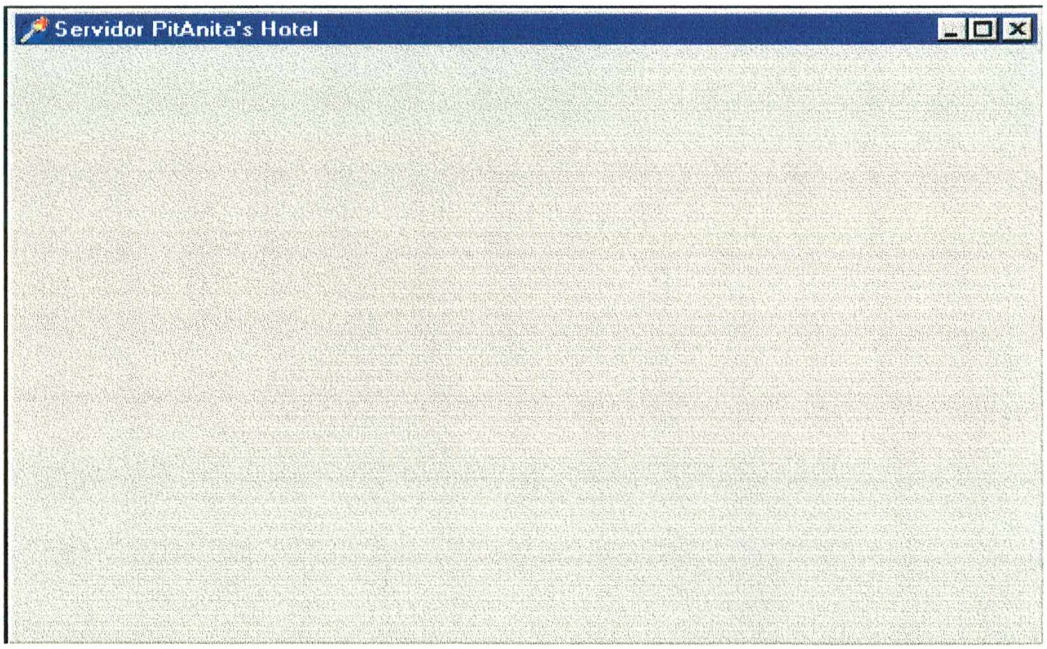

Figura 6.8 – Exemplo de um servidor Corba em execução

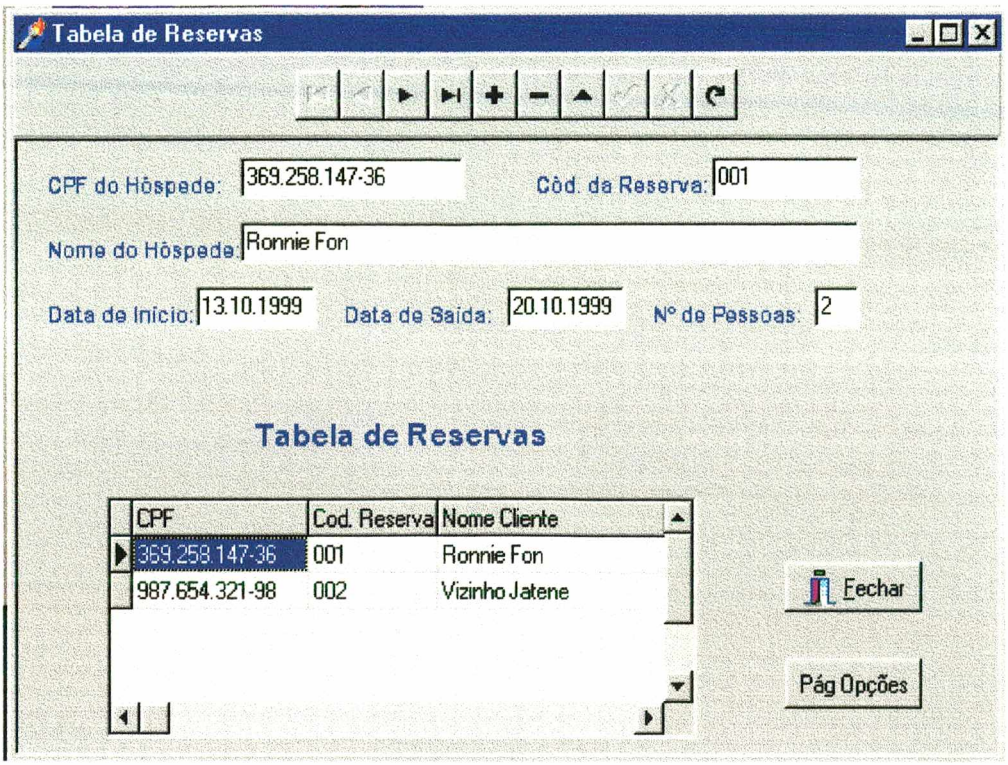

Figura 6.9 – Exemplo de um Cliente Corba em execução

6.2.6. SERVIDOR DCOM

Antes de começar a descrição do desenvolvimento do servidor Dcom, será feita uma breve explanação sobre o ***Microsoft Transaction Server*** (MTS), componente que será utilizado para o desenvolvimento deste servidor.

O **MTS** é um serviço de sistema operacional que pode ser instalado nas plataformas Windows. Ele é um ambiente de execução que fornece serviços de transação de bancos de dados, segurança, compartilhamento de recursos, e um aprimoramento em relação a robustez para os aplicativos Dcom. Os objetos gerenciados por ele, são objetos COM armazenados em um servidor em processo (uma DLL) e são manipulados pelo ambiente de execução MTS. Já os demais objetos COM são executados diretamente no aplicativo cliente.

Para aplicações de banco de dados, algumas das principais vantagens na utilização do MTS são:

- Segurança baseada em funções: permite determinar o direito de acesso de um cliente a interface de um módulo de dados, através de uma função dada ao mesmo;
- Recursos reduzidos de bancos de dados: existe uma camada intermediária entre o banco de dados e o servidor. Esta camada é conectada ao servidor e permite que as mesmas

conexões sejam utilizadas para diversos clientes. Isto faz com que o número de conexões de bancos de dados sejam reduzidas;

- Transações de bancos de dados: este serviço do MTS, suporta a conexão com vários tipos de bancos de dados.

A aplicação aqui desenvolvida utilizando a tecnologia Dcom é composta por dois servidores e um cliente, todos localizados em um único computador. A princípio, o objetivo era implementá-la da mesma forma que foi através do Corba. Contudo, foram encontrados alguns problemas durante a implementação, o que impediu a sua realização. Porém, a implementação de um cliente utilizando dois servidores simultaneamente foi possível e será demonstrada. Assim sendo, devido aos problemas encontrados, ao invés da estrutura da figura 6.1 que era a do objetivo inicial, teremos a seguinte estrutura:

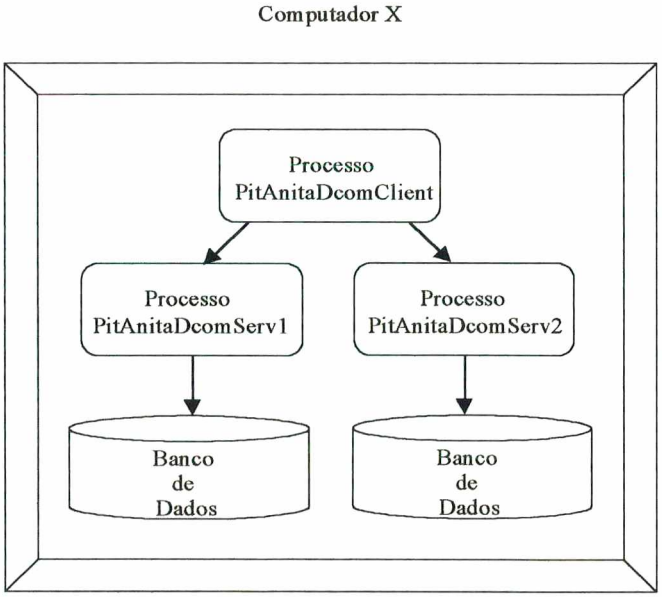

Figura 6.10 – Estrutura do PitAnita's Hotel sobre o DCOM

Para implementar o servidor deve-se criar um Módulo de Dados MTS, acrescentar a ele um componente *TTable*, conectar este componente ao BD e tabela devidos e exportá-lo. O tipo de biblioteca deve ser aberto de forma que se possa exportar o código do servidor do Pascal (código fonte do Delphi) para a IDL do Dcom. E por fim deve-se registrar o servidor.

No momento da criação de um Módulo da Dados MTS deverão ser selecionados os modelos de *threading* (fluxo de controle) e de transação.

Os modelos de fluxo de controle são:

- *Single*: o módulo de dados aceita apenas uma solicitação por vez. Todas as solicitações dos clientes são serializadas pelo MTS, de forma que o programador não precisa lidar com questões dos fluxos de controle.;
- *Apartment* ou *Single-threaded Apartment*: os métodos dos objetos só poderão ser acessados através do fluxo que ele foi criado. Poderão haver vários objetos de um mesmo servidor sendo chamados de fluxos diferentes, sendo que cada objeto apenas daquele fluxo específico;
- *Both*: os objetos suportam tanto os clientes que utilizam o *Apartment*, quanto os que utilizam o *Free* (ou *Mutli-threaded apartment*) - não disponível no Delphi. O *Free* permite que os métodos dos objetos sejam acessados através de qualquer fluxo.

E os modelos de transação são:

- *Requires a transaction* (Requer uma transação): toda chamada do cliente ao servidor é tida como uma transação MTS, a menos que o requisitor forneça um contexto de transação existente;
- *Requires a new transaction* (Requer uma nova transação): toda chamada é considerada uma nova transação MTS;
- *Supports transactions* (Suporta transações): o cliente deve fornecer um contexto de transação;
- *Does not support transactions* (Não suporta transações): o módulo de dados remoto não estará envolvido em nenhuma transação MTS.

A figura 6.11 mostra a janela da criação do módulo de dados MTS.

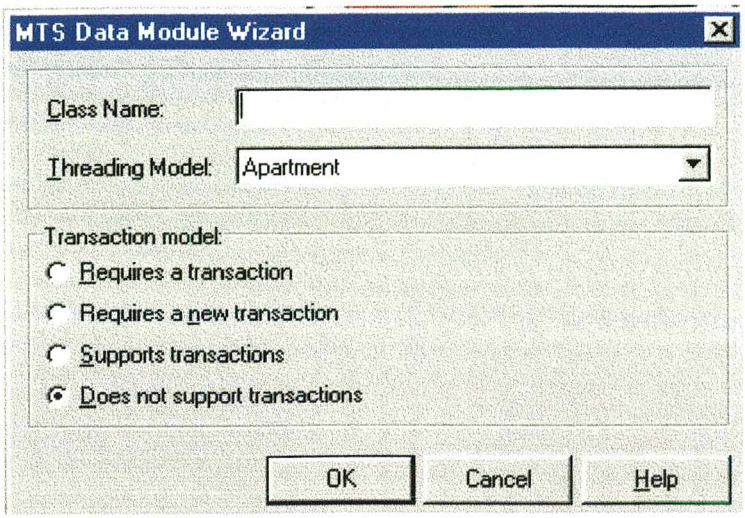

Figura 6.11 – Criando um Módulo de Dados MTS

É basicamente o mesmo processo do Corba, sendo que com componentes correspondentes. As principais diferenças em relação ao Corba é que este não precisa converter as interfaces para a sua IDL específica, pois elas já são escritas na mesma; não existe um repositório de interfaces para a adição de objetos; e o servidor deste não precisa ser executado, apenas registrado, inclusive na máquina cliente. Por esta razão também, não foi adicionado nenhum formulário, pois este só tem função quando o servidor é executado.

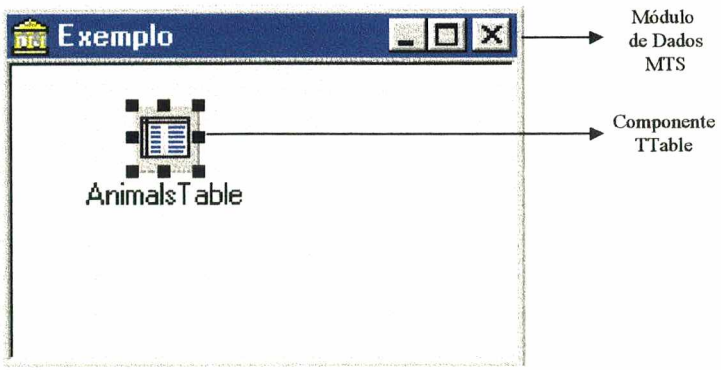

Figura 6.12 - Exemplo da Criação de um Servidor DCOM

Para a nossa aplicação foram criados dois servidores, o PitAnitaDcomServ1 e o PitAnitaDcomServ2. O Servidor 1 estabelece o acesso às tabelas de Lotação e Ocupação. E o Servidor 2, às tabelas de Reserva e Cadastro de Hóspedes. Ambos estão localizados na mesma máquina.

6.2.7. CLIENTE DCOM

O processo para a criação do cliente através do Dcom é o mesmo que o utilizado para o Corba, sendo que ao invés do componente *CorbaConnection*, será o *DcomConnection*. Assim como para o Corba era necessário especificar a propriedade do identificador de repositório, para o Dcom é preciso a do nome do servidor. Este é composto pelo nome do servidor mais o do módulo de dados.

Como o cliente da aplicação aqui desenvolvida se encontra local para ambos os servidores existentes, não há a necessidade de especificar o nome do computador em que o servidor está localizado. O restante do processo é todo idêntico ao da implementação local do Corba. Deve-se adicionar o *ClientDataSet*, o *DataSource* e os demais campos da tabela remota.

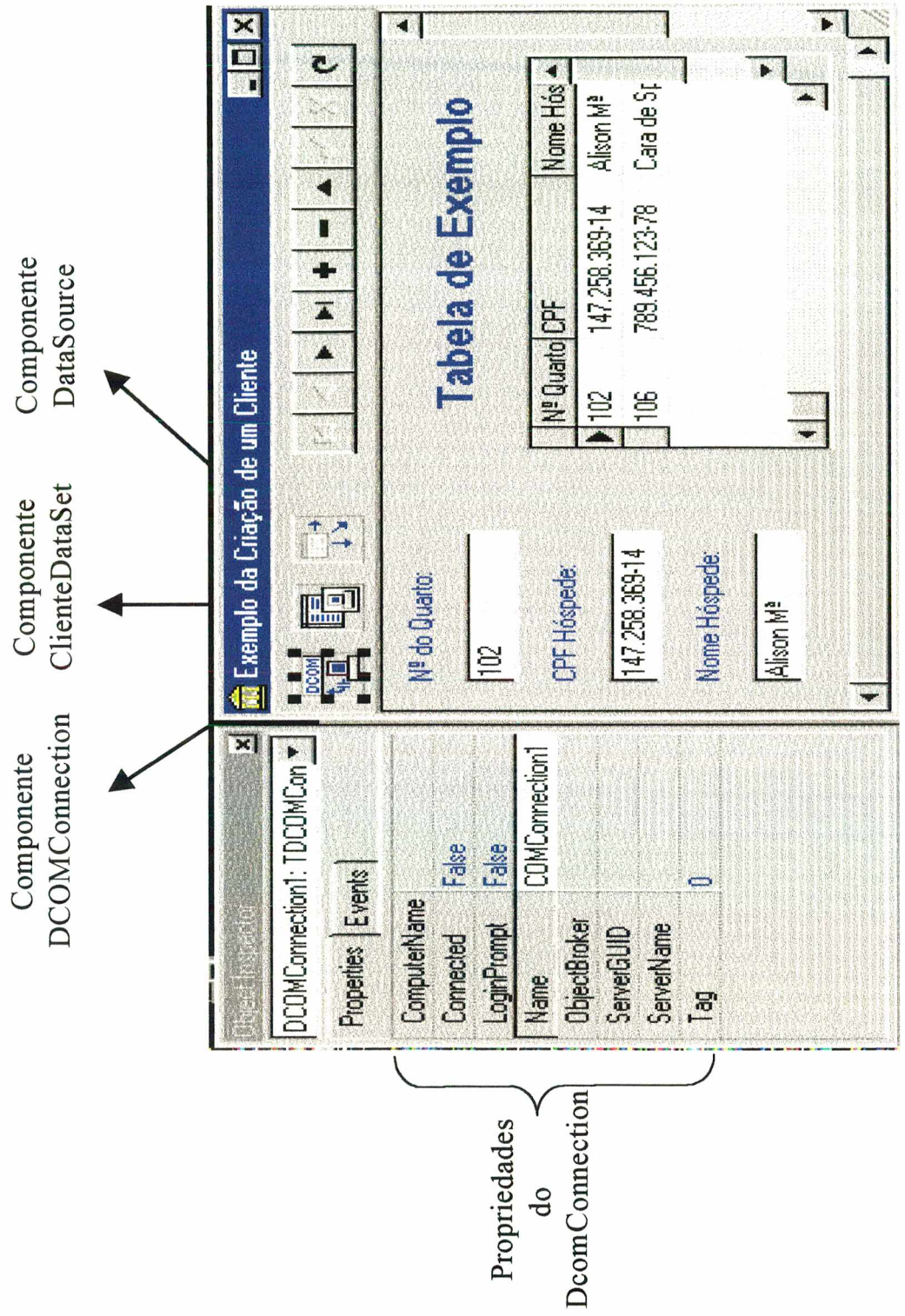

Figura 6.13 – Exemplo da Criação de um Cliente Dcom

A implementação do cliente e servidor Dcom, iniciou-se da mesma forma que se deu com o Corba, ou seja, com um único servidor e cliente localizados na mesma máquina. E assim como neste, foi obtido um resultado positivo.

Os mesmos testes com as diversas máquinas utilizadas para o Corba, também foram empregadas para testes com o Dcom. Sendo que os problemas para com este foram outros, os quais não foram solucionados.

O problema no desenvolvimento remoto do Dcom, é que neste para que um servidor seja acessado por um cliente localizado em outra máquina, ele (o servidor) deve ser registrado tanto na sua máquina quanto na cliente. Contudo, quando os testes eram realizados em máquinas diferentes, o registro do servidor não era possível.

Além disso existem também arquivos .dll que devem ser registrados, uns na máquina cliente e outros na servidora. E estes foram os obstáculos encontrados e motivo pelo qual somente foi possível apresentar resultados da implementação local. Uma lista de discussão também foi consultada para este caso, mas não foi obtido um resultado positivo.

Na estrutura final obtida existe um cliente que está localizado localmente para ambos os servidores. Este cliente corresponde aos funcionários do *PitAnita's Hotel* que possuem as mesmas funções que o PitAnitaCorbaClient1, que são a de efetuar registros e baixas de hóspedes, cadastrá-los e realizar reservas.

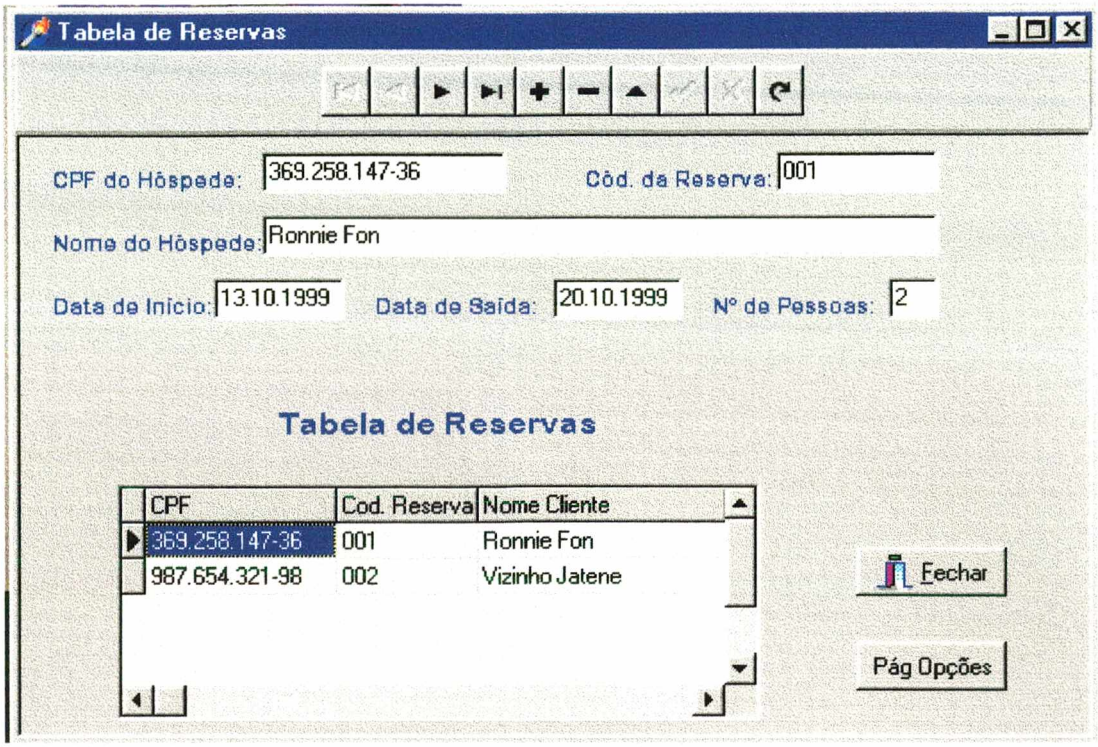

Figura 6.14 – Exemplo de um Cliente Dcom em Execução

6.3. Considerações Finais

O propósito deste trabalho era o de realizar um estudo sobre duas das tecnologias existentes atualmente para integração de ambientes heterogêneos , Corba e Dcom, de forma a se tornar um material de consulta para futuros trabalhos e pesquisas.

Visando tal objetivo foi realizado um estudo de cunho teórico sobre os componentes e funcionamento de cada uma destas tecnologias, essencial para o início do desenvolvimento que qualquer implementação na área, e também um levantamento bibliográfico comparativo que fornece dados e exemplos especificando as vantagens e dificuldades enfrentados por cada uma.

Por fim um experimento prático foi desenvolvido a fim de se obter uma visão real de tais tecnologias. O aplicativo proposto é o mesmo para ambas, de forma que se possa estabelecer uma comparação entre elas nos aspectos do desenvolvimento, recursos necessários e desempenho obtido.

VII CAPÍTULO
CONCLUSÃO

Foram apresentados, neste documento, os resultados dos estudos e experiências realizados visando a análise das duas propostas mais recentes de plataformas para o desenvolvimento de aplicações distribuídas orientadas a objetos, a saber, CORBA e DCOM.

7.1. SÍNTESE DOS RESULTADOS OBTIDOS

Para que se pudesse obter resultados significativos, a análise foi realizada em dois níveis: uma análise de cunho teórico, com base na documentação disponível; uma análise experimental, envolvendo uma implementação que utilizou como exemplo de reflexão uma aplicação na área de bancos de dados.

No primeiro estudo, pôde-se observar, uma maior abertura do Corba para o desenvolvimento de aplicações, uma vez que existem implementações desta plataforma disponíveis para as diversas arquiteturas e sistemas operacionais do mercado; já o Dcom, embora tenha as mesmas orientações, impõe uma arquitetura onde o servidor (ou servidores) do sistema devam ser baseados em Windows NT, o que, de certo modo, não foge à estratégia da Microsoft em fazer do seu sistema operacional a plataforma líder no desenvolvimento de sistemas corporativos.

No segundo estudo, algumas dificuldades foram, por outro lado, encontradas, desde a seleção da implementação até a obtenção de sucesso na operação da aplicação selecionada nos ambientes Corba e Dcom. Conforme relatado no capítulo 6, dentre as diversas opções possíveis em termos de ambiente de desenvolvimento, foi escolhido o ambiente Delphi (versão 4.0), o que foi determinado pela disponibilidade de uma versão que apresentava suporte ao desenvolvimento de aplicações em Corba e Dcom. Para realizar a análise, foi escolhido um exemplo simples de aplicação de base de dados, considerando basicamente dois perfis de implementação.

O primeiro perfil de implementação considerava todos os processos da aplicação localizados no mesmo posto de trabalho, esta decisão tendo sido tomada como estratégia de ajuste da aplicação no ambiente de desenvolvimento. O segundo perfil de implementação foi

projetado considerando os processos distribuídos ao longo de uma subrede dentro do ambiente computacional da rede do Departamento de Informática e de Estatística da Universidade Federal de Santa Catarina.

Para a obtenção de resultados desta segunda etapa de análise, os critérios considerados foram a facilidade de implementação, a adequação ao perfil pretendido da aplicação, a portabilidade e a taxa de sucesso na operação da aplicação.

No que diz respeito à facilidade de utilização, observou-se que este critério está mais vinculado ao ambiente de desenvolvimento adotado do que propriamente da proposta de arquitetura considerada. No caso deste estudo, o desenvolvimento da aplicação foi facilitado pela existência de componentes previamente construídos disponíveis no ambiente Delphi 4, o que tornou a implementação possível de ser realizada num prazo relativamente curto.

No tocante à adequação ao perfil da aplicação escolhida, no caso, uma aplicação de bancos de dados, também foi observado um nível de adequação bastante satisfatório no caso das duas plataformas, não tendo sido observadas diferenças significativas. É importante ressaltar, ainda, que o ambiente de desenvolvimento escolhido oferece facilidades importantes para o desenvolvimento de aplicação que fazem uso do modelo Cliente/Servidor, caso do exemplo escolhido no contexto deste trabalho.

No aspecto portabilidade, sem dúvida o desenvolvimento da aplicação sobre Corba apresentou melhores características, uma vez que ela oferece uma maior flexibilidade, principalmente no que diz respeito à localização do servidor ou servidores, o que não ocorre no caso de aplicações envolvendo o Dcom, que impõe o posicionamento de servidores em máquinas sob Windows NT, com o MTS instalado.

Finalmente, no aspecto sucesso operacional da aplicação, o Dcom, embora tenha permitido implementar rapidamente a aplicação com o perfil local (todos os processos localizados na mesma máquina), ofereceu dificuldades para o funcionamento no perfil distribuído, conforme descrito no capítulo 6.

Finalizando esta parte de apresentação de resultados, é importante ressaltar o interesse cada vez maior no desenvolvimento de aplicações distribuídas com base em plataformas voltadas a aplicações distribuídas orientadas a objetos, como Corba e Dcom. As vantagens da adoção de tal tecnologia são muitas, podendo-se resumir como principal benefício a

concentração dos esforços do desenvolvedor sobre as funcionalidades e requisitos da aplicação, podendo este abstrair-se dos problemas que podem surgir da distribuição dos processos e das eventuais conseqüências advindas da heterogeneidade do sistema computacional, uma vez que estas podem ser resolvidas pela adoção de uma tal plataforma.

7.2. PROPOSTAS PARA TRABALHOS FUTUROS

Embora este estudo tenha permitido um aprofundamento teórico e experimental dos conceitos envolvidos no uso das plataformas Corba e Dcom, é importante relacionar alguns pontos que podem ser abordados em futuros trabalhos relacionados a este tema, que por restrições, principalmente de tempo e por não estarem diretamente relacionadas ao objetivo maior deste trabalho não foram realizados. Estes aspectos são descritos a seguir.

7.2.1. Integração Corba x Dcom

Este é um estudo que pode apresentar uma contribuição importante na área, uma vez que o desenvolvimento de aplicações relacionando as duas plataformas pode permitir uma integração completa de processos clientes e servidores que tenham sido concebidos envolvendo Corba e Dcom. A dificuldade maior para o desenvolvimento de um tal tema será a obtenção de material sobre o assunto que permita conhecer detalhes de implementações disponíveis. No caso de Corba, o problema pode não ser tão significativo, uma vez que ele foi concebido dentro de um espírito de abertura, mas o mesmo não ocorre com a arquitetura Dcom que é de propriedade da Microsoft e que, como outros produtos da mesma empresa, não vai oferecer facilidades para a obtenção de informações técnicas relativas à sua implementação.

7.2.2. Desenvolvimento de Aplicações Distribuídas em Ambiente Heterogêneo

Outro ponto de fundamental importância é a análise experimental destas plataformas no contexto de um sistema computacional caracterizado por uma maior diversidade em termos de arquiteturas e sistemas operacionais. Embora este trabalho tenha tido o objetivo de realizar uma análise experimental de plataformas com esta orientação, diversos fatores dificultaram a criação de um ambiente mais propício a este tipo de análise.

7.2.3. Análise de Aplicações Distribuídas com Corba e Dcom com diferentes enfoques

Este é um outro ponto que não pôde ser encaminhado no contexto deste trabalho, mas que pode vir como complemento do mesmo no contexto de uma outra Dissertação de Mestrado. A idéia essencial de um trabalho deste gênero seria o de relacionar as propriedades comuns oferecidas nas duas plataformas, elaborando exemplos de aplicações que permitissem fazer uma avaliação mais detalhada destas propriedades, obtendo, quando possível, informações quantitativas, que podem ser mais úteis como referência num estudo comparativo.

REFERÊNCIA BIBLIOGRÁFICA

[1] BERTINI, Luciano. **Apresentação de Objetos Multimídia e Hipermídia em um Ambiente Distribuído Heterogêneo Utilizando os Padrões MHEG-5 e CORBA**. Florianópolis, 1998.

[2] BERSON, Alex. **Client/Server Architecture**. Singapore: McGraw-Hill, Inc., 1992.

[3] ORFALI, Robert; HARKEY, Dan. **Client/Server Programming with Java and Corba**. 2.ed.Canada: John Wiley & Sons Inc., 1998.

[4] SELTZER, Larry. **Components**. PC Magazine Brasil, v.9, n.1, pg.80, jan. 1999.

[5] FERNANDES, André; KANE, Roberto; ARCOVERDE, Rodrigo. **Delphi 4 Completo**. Rio de Janeiro: Book Express, 1998.

[6] SIMON, Errol. **Distributed Information Systems**. England: McGraw – Hill, 1996.

[7] LOFTUS, C.W. ... [et al]. **Distributed Software Engineering**. United States of America: Prentice Hall, 1995.

[8] CANTÚ, Marco. **Dominando o Delphi 4: "A Bíblia"**. São Paulo: Makron Books, 1998.

[9] CATTELL, R.G.G. **Object Data Management**. United States of America: Addison-Wesley Publishing Company, Inc.,1994.

[10] GRIETHUYSEN, J.J.. **Open Distributed Processing (ODP)**. Philips Technical Report. Eishoven (NL), 1990.

[11] TANENBAUM, Andrew S. **Sistemas Operacionais Modernos**. Holanda: Prentice-Hall, Inc., 1992.

[12] REISDORPH, Kent. **Teach Yourself Borland Delphi 4 in 21 Days**. June, 1998.

[13] SOUZA, Anamélia Contente. **Tecnologia para Ambientes Distribuídos – CORBA.** Monografia. Belém, 1998.

[14] ORFALI, Robert; HARKEY, Dan; EDWARDS, Jerl. **The Essential Distributed Objects Survival Guide**. Canada: John Wiley & Sons, Inc., 1996.

[15] JÚNIOR, Rivalino Matias. **Um Núcleo Multithreaded para Agentes de Gerenciamento OSI/ISO.** Monografia. Florianópolis, 1997.

[16] SAYEED, Imran. **Architecture: Objects, Middleware & Components. A Corba, DCOM Primer.** [on line]. Disponível na Internet via WWW: URL: http://www.adtmag.com/pub/jun98/fe601.htm

[17] BRANDO, Thomas J. **Comparing DCE and CORBA.** [on line]. Disponível na Internet via WWW: URL: http://www.mitre.org/research/domis/reports/DCEvCORBA.html

[18] **Component Object Model (COM), DCOM, and Related Capbilieties.** [on line]. Disponível na Internet via WWW: URL: http://www.sei.cmu.edu/str/descriptions/com_body.html

[19] TALLMAN, Owen; KAIN, J. Bradford. **COM versus CORBA: A Decision Framework.** [on line]. Disponível na Internet via WWW: URL: http://www.quoininc.com/quoininc/COM_CORBA.html

[20] CONDON, Ron. **CORBA no Match for Micorsoft's DCOM, Ovum Report Says.** [on line]. Disponível na Internet via WWW: URL: http://www.computerworld.com/home/online9697.nsf/all/970807corba

[21] **CORBA VS. DCOM.** [on line]. Disponível na Internet via WWW:URL: www.intraware.com/ms/itwr/snws/980605.html.

[22] HILL, Murray. **DCOM and CORBA Side by Side, Step by Step, and Layer by Layer.** [on line]. Disponível na Internel via WWW: URL: http://akpublic.research.att.com/ymwang/

[23] **DCOM Architecture White Paper.** [on line]. Disponível na Internet via WWW: URL: http://microsoft.com/ntserver/library/dcom_architecture.exe

[24] **DCOM Technical Overview.** [on line]. Disponível na Internet via WWW: URL: http://www.microsoft.com/ntserver/library/dcomtec.exe

[25] MCBRIDE, Simon. **DSTC RM-ODP Information Service.** [on line]. Disponível na Internet via WWW: URL: http://dstc.edu.au/AU/research_news/odp/rel_model/rel_model.html

[26] **For CORBA and DCOM it's Time to Get Practical.** [on line]. Disponível na Internet via WWW: URL: http://www.byte.com/art/9704/sec8/art1.htm

[27] **OMG Workshop on the Object Management Architecture and The Reference Model for Open Distributed Processing (RM-ODP).** [on line]. Disponível na Internet via WWW: URL: http://enterprise.Systemhouse.MCI.com/odp-oma/odp-oma.html

[28] **Open Distributed Processing.** [on line]. Disponível na Internet via WWW: URL: http://enterprise.Systemhouse.MCI.com/WG//default.htm

[29] **OSF DCE RPC & DCOM Protocol.** [on line]. Disponível na Internet via WWW: URL: http://iwt.npac.syr.edu/people/rsshitol/dcomppt/tsld004.htm

[30] **RM-ODP and URI.** [on line]. Disponível na Internet via WWW: URL: http://www.base.com/gordori/web/rees-odp.html

[31] **The Open Force vs. The Legacy Object.** [on line]. Disponível na Internet via WWW: URL: http://www.byte.com/art/9707/sec10/art2.htm

[32] **The Open Group Portal to the World of DCE.** [on line]. Disponível na Internet via WWW: URL: http://www.opengroup.org/dce